VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hra Dáma

The Game Checkers

2013 Roland Krutek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Roland Krutek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Hra Dáma
The Game Checkers

Zásady pro vypracování:

Cílem je vytvořit funkční aplikaci hry Dáma s různými úrovněmi obtížnosti - 2 hráči proti sobě, hráč vs. počítač - jednoduchá verze, obtížná verze. Dále by student měl nastudovat evoluční algoritmy a aplikovat na tuto hru.

1. Nastudovat problematiku logické počítačové hry Dáma.
2. Implementovat hru s těmito zásadami:
a) grafické uživatelské rozhraní,
b) zamezit nepovoleným tahům,
c) nápověda, který tah by měl být ten správný,
d) různé obtížnosti hry,
e) dva hráči vs. hráč a počítač.
3. Nastudovat rodinu evolučních algoritmů a aplikovat přímo ve hře.
4. Shrnutí dosažených výsledků.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Lenka Skanderová**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 29. dubna 2013 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem této práce bylo nastudovat problematiku logické počítačové hry dáma a způsoby řešení umělé inteligence pro tuto hru. Výsledkem je program s grafickým uživatelským rozhraním a s více úrovněmi a variantami umělé inteligence. Dále se práce zabývá možností aplikování evolučních algoritmů ve hře dáma.

Klíčová slova: dáma, umělá inteligence, naivní algoritmus, Minimax, Negamax, alfa-beta ořezávání, evoluční algoritmy, genetický algoritmus, diplomová práce

Abstract

The aim of this thesis was to study the issue of logical computer game of checkers and ways of creating artificial intelligence for this game. The result is a program with a graphical user interface with multiple types and levels of artificial intelligence. This thesis also deals with the possibilities of applying evolutionary algorithms in the game of checkers.

Keywords: checkers, draughts, naive algorithm, Minimax, Negamax, alfa-beta pruning, evolutionary algorithms, genetic algorithm, artificial intelligence, master thesis

Seznam použitých zkratek a symbolů

UI	–	Umělá inteligence
EA	–	Evoluční algoritmus
GA	–	Genetický algoritmus
ACO	–	Ant Colony Optimisation (Optimalizace mravenčí kolonií)
PSO	–	Particle Swarm Optimization (Optimalizace rojením částic)
SOMA	–	Self-Organising Migrating Algorithm (Samoorganizující se migrační algoritmus)
DE	–	Diferential evolution (Diferenciální evoluce)

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Dáma je logická desková hra, která se těší velké přízni fanoušků z celého světa. V České republice jde o nejhranější deskovou hru a v žebříčku popularity předběhla i šachy, které se nachází aktuálně na druhé pozici (viz. [1]). Tím, že ve hře nehraje žádnou roli náhoda, ale o výsledku partičky rozhodují výhradně vědomosti hráčů, se hra stala zajímavou i z hlediska vývoje umělé inteligence (UI). Dnes existují dva základní přístupy k vytváření umělé inteligence ([2] a [3]). Jedním z nich je „naučit“ stroj všemu, co potřebuje vědět k vykonání činnosti, ke které je určen. Druhým je nechat stroj, aby se učil sám.

První možnost spočívá v předání různých informací stroji, který pomocí těchto nabytých informací dokáže vyřešit určitý specifický úkol. Programátoři v tomto případě transformují své vědomosti do podoby, které stroj rozumí. Všechny informace, které stroj potřebuje například k hraní hry Dáma, jsou mu předem poskytnuty.

Druhá možnost je vytvořit algoritmus tak, aby byl sám schopen učit se. Touto možností se zabývá tzv. evoluční programování, které se inspirovalo biologickou evolucí. Tento přístup přímo neřeší daný problém, ale snaží se k řešení postupně dopracovat. Nástrojem k nalezení optimálního řešení jsou tzv. evoluční algoritmy (EA).

2 Hra Dáma

Hra, kterou se tato diplomová práce zabývá, je celosvětově známá. Její jméno se liší v závislosti na zemi, ve které ji hrají. V Německu nese název *das Damenspiel*, v Itálii *Dama* a Francouzi ji znají pod jménem *La Jeu de Dames*. V Česku se ji říká Dáma a už samotné pojmenování hry nám napovídá, že se jedná o vznešenou hru. O hru, kterou kdysi hráli nejen evropští králové, ale možná také egyptští faraóni. Jestli je tomu skutečně tak, to přesně nikdo neví. Původ hry je zahalen rouškou tajemna a v současné době můžeme jen hádat, odkud se ve skutečnosti vzala.

2.1 Počátky a historie hry Dáma

Existuje mnoho názorů na to, odkud hra pochází a kdo je jejím autorem. Podobných her nalezneme desítky, možná i stovky. Jsou to například hry: *Adugo, Komikan, Rimau, Sher-bakar*, atd.

Popis těchto her, ale také mnoha dalších, můžeme nalézt například v díle Miloše Zapletala, *Velká kniha deskových her* [4].

Některé z nich jsou Dámě více podobné, jiné se jí podobají méně. Která z těchto her, jestli vůbec některá, je jejím předchůdcem, to netušíme. Badatelé i historici, kteří se touto problematikou zabývají, tvrdí, že mezi nejžhavější kandidáty patří Senet, Ludus latrunculi, Petteja, nebo Alquerque.

2.1.1 Senet

Podle některých zdrojů (např. [5]) sahají kořeny Dámy až do doby 3 000 př. n. l. Jejím předchůdcem by mohla být hra, kterou Staroegypťané nazývali Senet. O původu této hry vznikly různé legendy. Autorem jedné z nich je řecký filosof Platón. Podle něj je Senet výmyslem egyptského Boha moudrosti, zvaného Thovt, který ji hrál s Bohyní měsíce a dokázal nad ní vyhrát pouze jedinkrát za 4 roky. Tuto „božskou partičku“ zachycují malby v hrobkách, nalezené právě v Egyptě (Obrázek 1). Je na nich znázorněna hra, která se Dámě do jisté míry podobá. Nalezneme zde ale zásadní rozdíly, které přináší jisté pochybnosti o provázanosti těchto dvou her.

I když o přesných pravidlech Senetu toho moc nevíme, jisté je, že se hraje s počítacími paličkami, které určují, o kolik pozic se hráč může se svou figurkou posunout. Stejný způsob rozhodování se používá i ve hře dnes známé jako Vrhcáby (Backgamon), ale zde byly počítací paličky nahrazeny hracími kostkami. Senet je tedy hra, ve které hraje důležitou roli štěstí, zatímco o Dámě jsme si už v úvodu řekli, že je to hra čistě o vědomostech hráčů, nikoliv o náhodě. Podoba hrací desky připomíná spíše hrací desku Backgamonu. Také způsob, kterým lze hru vyhrát, je téměř identický u obou her. Pro výhru je nutné své figurky dostat na jistou pozici. Z této pozice pak figurky opouští hru. Lze tedy předpokládat, že Senet je spíše předchůdcem dnešního Backgammonu, než Dámy. To ale nevylučuje možnost, že se Dáma odvíjela právě od hry Senet.

Obrázek 1: Freska znázorňující hru Senet [6]

2.1.2 Ludus latrunculi

Jiná legenda říká, že začátky Dámy lze datovat na období přibližně 1 200 př. n. l. V té době mělo stát na území dnešního Turecka bájné město Trója. Hra zřejmě vznikla při obléhání města řeckými vojáky. Mezi obléhateli byl i achájský vynálezce a hrdina Palamédés. Podle Homéra je právě on autorem deskové hry, připomínající Dámu. Nalezly se i kresby (Obrázek 2) na starověkých keramických nádobách, na kterých je vyobrazen postava dalšího z řeckých hrdinů Achillese, jak si hrou krátí čekaní před hradbami.

Hra se jmenuje Ludus latrunculi (známá také jako Ludus latrunculorum) a Řekové ji vymysleli pro zdokonalování svých dovedností v oblasti vojenské taktiky. Hra se už pravidly více podobá dnešní Dámě jako Senet. Popis i pravidla můžeme nalézt např. v [4].

2.1.3 Petteia

Podle historiků je Ludus latrunculi jakousi rozšířenou verzí jiné řecké deskové hry zvané Petteia, Pessoi nebo také Polis. Její název se liší podle kraje, ve kterém ji hráli. Hrací deska měla 8 x 8 políček a figurky se mohly pohybovat jen směrem dopředu. Hru popsal starořecký historik Hérodotos v několika svých příbězích. O jejích pravidlech se můžeme dočíst v kodexu Alfonse X. Moudrého, pocházejícího ze 14. století nebo v [4].

2.1.4 Alquerque

Harold J. R. Murray prohlásil za nejpravděpodobnějšího předchůdce Dámy hru s názvem Alquerque. Tohle tvrzení zdůvodnil ve své knize o historii deskových her [8]. Kořeny Dámy by podle něj měly být ve Francii. Jistí si ale být nemůžeme, protože hru připomínající Alquerque popsal autor Abu al-Faraj al-Isfahani ve své knize *Kitab al-Aghani* („Kniha písní“), již v desátém století. Pravidla hry ale nezmínil.

Obrázek 2: Partička Ludus lanctruculi zobrazena na řecké amfoře [7]

2.1.5 Dáma v Česku

Nejspíše se nikdy nedozvíme, kdo byl autorem hry a kde vznikla. Jisté je, že do Česka se Dáma dostala v 17. století ze Španělska a Německa. Zasloužil se o ni Karel II., poslední král z rodu Habsburků, zvaný také Karel Španělský. Z tohoto důvodu se pravidla nejvíce podobají právě těm španělským a německým. I samotný název hry je z německého der Damm.

Hra se rychle stala populární a oblíbenou. Hrála ji jak šlechta, tak i prostí lidé. Pravidla Dámy nikde nebyla přesně definována, proto ji různí lidé hráli podle různých pravidel. První pokus o prosazení jednotných pravidel pro celé Česko vznikl až ve 20. století. Tehdy se v Česku začala hrát tzv. Mezinárodní dáma, díky které se otevřely dveře na mezinárodní turnaje. Pravidla České dámy unifikovala ČESKÁ FEDERACE DÁMY, která vznikla až v roce 1993.

Česko se tak připojilo mezi rozsáhlou množinu krajin, které kromě mezinárodních pravidel, mají i svá lokální pravidla, platná jen pro danou zemi. Dnes si už prakticky každý národ pravidla přizpůsobil, a proto existuje mnoho variant hry [4].

V další sekci se budu věnovat některým z nich, která považuji z hlediska této diplomové práce za důležitá.

2.2 Varianty hry

Vybrané variace, které si popíšeme jsou: Mezinárodní dáma, která se hraje na mezinárodních turnajích, Dáma podle českých pravidel a anglická verze hry.

2.2.1 Mezinárodní dáma

Česká federace dámy definuje na svých webových stránkách hru následovně: „*Česká dáma je duševní sport, který hrají dvě osoby, které se nazývají hráči.*“ [1] Hra se řídí následujícími pravidly:

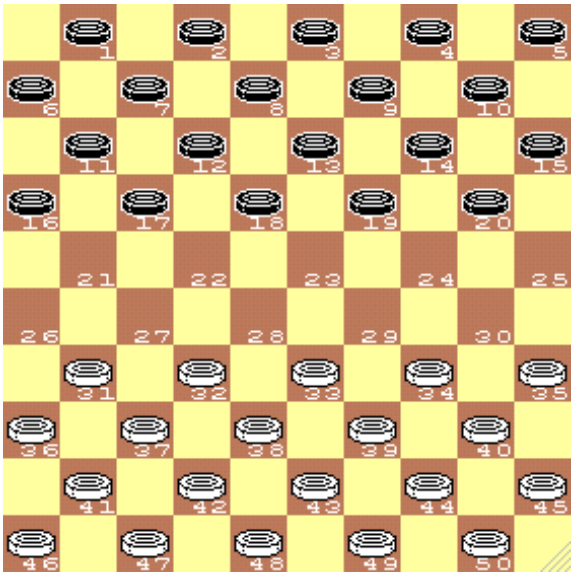
- Hraje se na desce 10x10, na každé straně 20 kamenů ve 4 řadách na černých polích.
- Kameny se táhne po diagonálách, vždy jen o jedno pole dopředu.
- Skáče se dopředu i dozadu, hráč přenese svou figuru na konečné pole skoku, naznačuje přitom průběh skákání a až poté odstraní přeskočené figury z desky.
- Přes jeden kámen lze skákat jen jednou.
- Skákání je povinné. Opomenuté skákání se promlčuje dalším tahem soupeře.
- Přednost má ten skok, kterým se přeskakuje větší množství soupeřových figur.
- Dokončí-li kámen tah na poslední řadě (bílý na polích 1-5, černý na polích 46-50), mění se v dámu.
- Dáma se pohybuje po diagonálách dopředu i dozadu, může skákat přes libovolný kámen na diagonále a dokončit skok na kterémkoli poli za přeskočeným kamenem.
- Dáma nemá při skákání přednost.
- Partii vyhrává hráč, jehož soupeř nemá tah podle pravidel, tzn. že nemá již žádné figury, nebo jsou jeho figury zablokovány, nebo se vzdal.
- Partie skončí nerozhodně dohodou hráčů, nebo když se vyskytne 3x stejná pozice.

Hrací deska a implicitní rozmístění figurek je zobrazeno na Obrázku 3.

2.2.2 Česká Dáma

Pravidla české dámy také nalezneme na oficiálních stránkách české federace Dámy [1].

- Hraje se na desce $8x8$, na každé straně 12 kamenů ve 3 řadách na černých polích.
- Kameny se táhne po diagonálách, vždy o jedno pole dopředu.
- Dokončí-li kámen tah na poslední řadě (bílý $b8, d8, f8, h8$, černý $a1, c1, e1, g1$), mění se v dámu.
- Dáma se pohybuje po diagonálách dopředu i dozadu.
- Kameny se skáče jen dopředu, Dáma může skákat přes libovolný kámen na diagonále dopředu i dozadu a dokončit skok na kterémkoli poli za přeskočeným kamenem.



Obrázek 3: Výchozí pozice pro mezinárodní dámu a číslování desky

- Hráč při skoku přenese svou figuru na konečné pole skoku, naznačuje přitom průběh skoku a až poté odstraní přeskočené figury z desky. Přes jeden kámen lze skákat jen jednou.
- Skákání je povinné a figura po dokončení tahu nesmí mít další možnost skoku. Je-li více možností skákání, může si hráč vybrat bez ohledu na množství přeskočených kamenů soupeře. Opomenuté skákání se promlčuje dalším tahem soupeře.
- Dáma má při skákání vždy přednost před kamenem.
- Partii vyhrává hráč, jehož soupeř nemá tah podle pravidel, tzn. že nemá již žádné figury, nebo jsou jeho figury zablokovány, nebo se vzdal.
- Partie skončí nerozhodně dohodou hráčů, nebo když se vyskytne $3x$ stejná pozice.

Česká verze se od mezinárodní liší hlavně velikostí hrací desky, které dimenze jsou v obou směrech menší o dvě políčka. Počet figurek se také zredukoval, a to z dvacet na dvanáct, přičemž figurky, které se nestaly králem, se mohou pohybovat jen směrem dopředu.

2.2.3 Anglická dáma

Anglická dáma (draughts) je populární prakticky ve všech anglicky mluvících zemích jako jsou Velká Británie, Kanada nebo Spojené státy americké. Anglická dáma má svými pravidly blíže spíš k české variantě než mezinárodní. To je jeden z důvodu, proč se jí zabývají i čeští autoři. Jedním z nich je např. již zmíněný M. Zapletal, který ve své knize [4] sepsal pravidla Anglické dámy. Rozdíly mezi anglickou a českou verzí jsou:

- Pro označení figurky, která se dostala na jednu ze soupeřových krajních pozic se používá pojmenování král.
- Pokud hráč může provést tah jak králem, tak i obyčejnou figurkou, může se rozhodnout, kterou figurkou potáhne.
- Král se na rozdíl od dámy z české varianty nemůže pohybovat o libovolný počet políček. Pohybovat se může vždy jen o jedno políčko, stejně jako obyčejné figurky, ale všemi směry.

Anglickou dámu jsem si vybral jako základ pro svůj program z toho důvodu, že svými pravidly se hra nejvíce podobá Dámě, kterou jsem v dětství hrával, tudíž je mi nejblíž. Navíc, pro českého hráče je přirozenější hrát hru podle anglických pravidel, než podle pravidel mezinárodních.

Co se implementace týče, je naprogramování kterékoliv varianty stejně náročné. Pro náš program si některá pravidla mírně přizpůsobíme. Jedno z těchto pravidel nám říká, že hráč musí vykonat skok v případě, když nějaký existuje. Může ale nastat situace, kdy si ani jeden z hráčů nevšimne, že tato možnost existovala. Takové opomenuté skákání se promlčuje dalším tahem soupeře. V našem případě ke skoku hráče můžeme donutit znemožněním jakéhokoliv jiného tahu.

Remíza ve hře nenastává po dohodě hráčů s UI. Remíza nastane po překročení jistého limitu pro počet tahů partičky.

2.3 Umělá inteligence

P. McCorducková ve své knize [9] popsala UI jako starodávnou touhu o napodobení Bohů. Z těchto slov si můžeme odvodit, že UI není jen tématem moderní doby, ale že se myšlenkou zabývali už naši předkové. Nicméně její abstraktní definice nám nedává jasnou představu o tom, co to vlastně UI je. Proto než se začneme tématem hlouběji zabývat, je důležité si říci, co vlastně pod pojmem UI máme na mysli.

Běžně se inteligence definuje jako rozumová schopnost řešit nově vzniklé nebo obtížné situace, učit se ze zkušeností, přizpůsobit se, určit podstatné vztahy a souvislosti. Bez pochyb jde o vlastnost, která je typická pro člověka. V případě, že tuto vlastnost dokážeme přenést na nějaký stroj, vytváříme tím UI. Kdy ale můžeme prohlásit, že je stroj inteligentní?

Tímto tématem se zabýval britský matematik Alan Turing [10], který popsal způsob, jak je podle něj možné rozhodnout, zda-li je stroj inteligentní, či nikoliv. K testu potřebujeme tři účastníky:

- testera (osoba, která se pokouší rozhodnout o inteligenci stroje),
- testovaný stroj, o kterém chceme rozhodnout, zda-li je inteligentní,
- testovanou osobu.

Princip testu je následující. Aktéři obsadí oddělené místnosti. Tester nesmí vědět, který z aktérů je ve které místnosti. Začne pokládat otázky jak testované osobě, tak i testovanému stroji. Stroj i testovaná osoba na otázky odpovídají. Pokud tester není schopen rozeznat, kdy odpovídá stroj a kdy člověk, považuje se stroj za inteligentní. Tzn., že pokud je stroj po lingvistické stránce na stejné úrovni jako člověk, považuje se za inteligentní. Tento test nese název svého autora a říká se mu Turingův test.

Správnost tohoto testu zpochybňuje tzv. argument čínského pokoje. Ten říká, že by teoreticky mohl existovat stroj, který by na každou otázku měl předem připravenou odpověď. To by vedlo k tomu, že by tester nedokázal nalézt lingvistický rozdíl mezi strojem a člověkem. Stroj by se vyhlásil za inteligentní navzdory tomu, že ve skutečnosti žádné známky inteligence nevykazuje. Samozřejmě takový stroj ve skutečnosti existovat nemůže, protože nejsme schopni předprogramovat odpovědi na všechny možné otázky.

2.3.1 Historie UI

Motiv uměle vytvořených bytostí obdařených inteligencí a vědomostmi můžeme nalézt prakticky snad ve všech kulturách po celém světě. Už ve starověkém Egyptě vznikaly mýty o podobných stvořeních. V řecké mytologii je to například obrovský humanoidní robot Talós, vytvořený Bohem kovářství Héfaistem. V moderní době je to Frankenstein, uměle vytvořený člověk z hororového románu Mary Shelleyové vydaného v roce 1818. Tudíž můžeme učinit prohlášení, že snaha o předání svých vědomostí a lidských vlastností živým i neživým objektům je v povědomí lidstva přítomna už od pradávna, až po dnešní dobu. [11]

Na odbornější úrovni se tématem začalo lidstvo zabývat v době přibližně 500 až 300 př. n. l., kdy se skupinka několika filozofů pokusila namodelovat procesy lidského uvažování pomocí symbolu. Tím vlastně položili základní kámen moderní podoby UI. Ta zažila svůj rozmach až od roku 1940, kdy uzřel světlo světa první programovatelný digitální počítač. Ten inspiroval vědce k tomu, aby se seriózně začali zabývat myšlenkou stvoření elektronické podoby mozku. O tři roky později skupina amerických vědců v čele s Warrenem McCullockem a Walterem Pittsem vydala vůbec první publikaci zabývající se tématem elektronického mozku. Jejich publikace veřejnost natolik zaujala, že jim univerzita povolila výzkum UI přímo na své půdě. Jejich dlouhotrvající snažení vyústilo ve vytvoření počítače, který se snažil simulovat inteligenci pomocí propojených neuronových sítí. Bohužel takto vytvořený počítač nevykazoval žádné známky inteligence. Počáteční euforie a velké naděje vystřídalo zklamání jak vědeckého obecenstva, tak i široké veřejnosti. Tím pádem se výzkum na poli UI na dlouhá léta prakticky úplně zastavil. Až v 90. letech minulého století, kdy výpočetní síla počítačů několikanásobně vzrostla, se téma UI opět dostalo do popředí. Začaly vznikat expertní systémy, tj. počítače, které na základě simulovaných analytických schopností dokázaly nalézt řešení daného problému. Tato událost opět nastartovala vývoj UI, který pokračuje až dodnes. [11] [12]

V dnešní době se už UI využívá například v matematice, lékařství, geologii, právu nebo dokonce i ve hrách. Právě využitím UI ve hrách se zabývá další podkapitola.

2.3.2 UI a hry

Hry se už od počátku moderní éry UI používají jako měřítko pokroku v oblasti UI. Z toho důvodu má smysl se jimi zabývat. Nás budou zajímat hlavně deskové hry jako Šachy, Dáma, Mlýn a podobně. V roce 1951 vytvořili studenti University of Manchester první počítačové hry, Dámu a Šachy, které měly svou vlastní UI. Inteligence těchto programů ale ještě nebyla nijak na vysoké úrovni. Až o desetiletí později se povedlo Arthurovi Samuelovi vytvořit UI pro hru Dáma, která už byla schopna porazit zkušenější hráče. Jeho UI navíc byla schopna učit se z předešlých zkušeností.

Dnešní UI jsou už na tak vysoké úrovni, že jsou schopné porážet i největší hráče. Za zmínku stojí například nejznámější šachový program nazvaný Deep Blue, který v roce 1997 porazil tehdejšího mistra světa Garryho Kasparova. Fritz nebo Schredder jsou taktéž šachové programy, které jsou na úrovni, která hraničí s neporazitelností.

Z jiné kategorie pochází Watson. Ten je dalším programem, který byl schopen porazit své lidské oponenty ve vědomostní televizní soutěži Jeopardy! (v Česku známe pod názvem Riskuj!). V Dámě je to pak program Chinook. Chinook je prvním počítačovým programem, který získal titul mistra světa v soutěži proti lidským protihráčům. UI Chinooku se pokládá za neporazitelnou. Program dokázal, že pokud Dámu hrají hráči dokonale, tj. neudělají žádnou chybu, tak hra vždy končí remízou. Chinook „vyřešil“ Dámu a to ze všech možných pozic na 8x8 šachovnici. Využil k tomu algoritmu založeném na tzv. brute force, tj. hrubé síle. Podstatou algoritmu zmíněného typu je nalezení všech možných řešení a z množiny těchto řešení vybrat tu nejvhodnější pro danou situaci. Mezi algoritmy založených na hrubé síle patří i algoritmus Minimax, nebo tzv. naivní algoritmus. [13]

Na stejných algoritmech bude založen náš program. Důležité je ale připomenout, že cílem této diplomové práce v žádném případě není vytvořit neporazitelnou UI, ale ukázat klasický i moderní přistup pro vytvoření UI pro hru Dáma a porovnat je.

3 Vybrané algoritmy

Než se na vybrané algoritmy podíváme z blízka, je nutné si představit datovou strukturu zvanou strom, která se ve zmíněných algoritmech využívá.

3.1 Stromy

Definice 3.1 *Strom je souvislý graf neobsahující kružnici. [14]*

Stromem v informatice (ale také v matematice) nazýváme strukturu, která svým grafickým zobrazením připomíná strom, jak ho známe z přírody. Z definice 3.1 vyplývá, že nemůže nastat situace, kdy mezi dvěma hranami bude víc než jedna cesta. Z neexistence kružnice vyplývá, že do uzlu A vede z uzlu B právě jedna cesta. Tím, že se jedná o souvislý graf, můžeme o něm prohlásit, že z libovolného uzlu A musí vést cesta do libovolného uzlu B.

Ve stromu může každý z uzlů mít potomka. K potomkovi vede vždy jedna hrana. Pokud uzel nemá potomka, říkáme, že je to list stromu. Pokud naopak uzel nemá předka, nazýváme uzel kořenem stromu. Listy stromu zobrazeného na Obrázku 4 jsou uzly H, E, I a G. Kořenem tohoto stromu je uzel A.

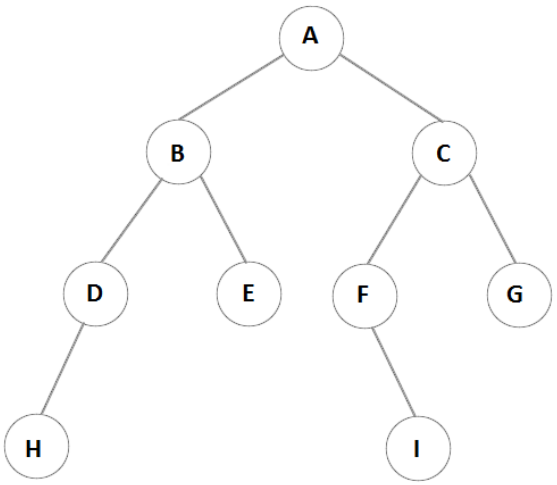
Stromy se v informatice využívají například pro ukládání různých dat. Jedním z variant stromů vhodných pro ukládání dat je například binární vyhledávací strom (BST z anglického Binary Search Tree).

BST je strom, ve kterém může mít každý z uzlů nejvíce dva potomky. Proto nazýváme strom binárním. Nemůže nastat situace, kdy z jednoho uzlu vedou více než tři hrany k jiným uzlům. Dva uzly mohou vést k potomkům a jeden k předkovi. Pravidlem přitom je, že levý potomek by měl obsahovat hodnotu, která je větší nebo rovna svému předkovi. V pravém se nachází hodnota větší. Takto vytvořená stromová struktura umožňuje jednoduše vyhledávat hodnoty. Při navštívení libovolného uzlu stačí porovnat, jestli je hodnota větší nebo menší hodnotě nacházející se v navštíveném uzlu. Když nastane situace, kdy je hodnota stejná, hodnotu jsme nalezli a procházení stromu ukončíme. Pokud je hodnota větší, pokračujeme vpravo, pokud menší, pokračujeme vlevo. Pokud je rovna, tak jsme prvek nalezli. Může se stát, že při prohledávání stromu dosáhneme maximální hloubky stromu. V tom případě algoritmus skončí v uzlu definovaném jako list. V případě, že tato situace nastane a prvek dosud nebyl nalezen, nenachází se ve stromu [14].

Vyhledávání ve stromech je mnohem rychlejší, než vyhledávání v polích nebo v seznamu. To je hlavním důvodem, proč se v informatice často využívají.

3.2 Naivní algoritmus

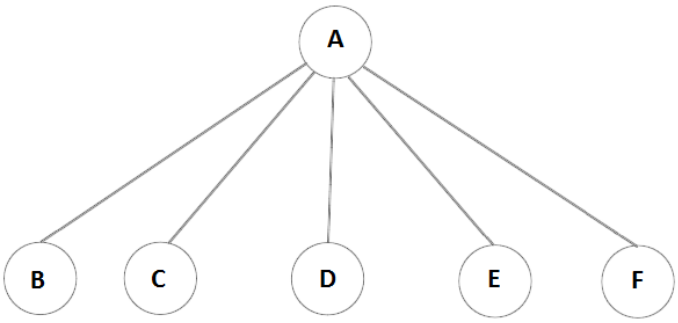
Jedná se o nejjednodušší algoritmus pro vytvoření agenta UI. Avšak jeho jednoduchost je na úkor inteligence. Jde o algoritmus, který vybere nejlepší možný tah z tahů, které lze provést z dané situace. Algoritmus uvažuje přesně jeden krok dopředu, tj. má hloubku 1. Proto můžeme s troškou nadsázky říci, že se jedná o variantu algoritmu minimax s hloubkou 1, o kterém si řekneme v další sekci. Algoritmus je popsaný pseudokódem ve



Obrázek 4: Stromová datová struktura

Výpisu 1. Algoritmus vytváří stromovou strukturu hloubky 1, kterou znázorňuje Obrázek 5. Kořen stromu, uzel A, je výchozí pozice, ve které se UI nachází. Z této pozice se pak může provedením jednotlivých tahů dostat do uzlů B až F. Algoritmus pomocí vyhodnocovací funkce, kterou se budeme zabývat později, vyhodnotí každý tah a přiřadí ji číselnou hodnotu, která zrcadlí její výhodnost. Nakonec se vybere tah, který se jeví jako nejlepší. [15]

Síla nejlepších hráčů deskových her světa spočívá v tom, kolik tahů dopředu dokážou myslet. Proto nemůže naivní algoritmus konkurovat algoritmům, o kterých si budeme povídat v další části a už vůbec ne lidským soupeřům.



Obrázek 5: Stromová struktura naivního algoritmu

```
public Move NaiveAlg(Stav s, Hrac hrac){
    nejlepsiSkore = −NEKONECNO;
    if (jeMozneVykonatSkok())       /* Pokud existuje dostupný skok, musí být upřednostněn */
    {
        vsechnyMozneTahy = GenerujSkoky(hrac);    /* Vrátí všechny skoky hráče */
    }
    else
    {
        vsechnyMozneTahy = GenerujOstatniTahy(hrac);    /* Vrátí všechny ostatní tahy hráče */
    }
    /* Pokud neexistuje platný tah, vrátím null a hra skončí výhrou soupeře */
    if (vsechnyMozneTahy.pocet == 0)
    {
        return null;
    }
    for (i = 0; i < vsechnyMozneTahy.pocet; i++)      /* Iterace skrze všechny tahy */
    {
        ProvedTah(s, vsechnyMozneTahy[i]);        /* Zahrání tahu */
        skoreTahu = Ohodnot(stavPoTahu);      /* Vyhodnocení tahu */

        /* Je−li tah lepší než dosavadní nejlepší, bude nový nejlepší */
        if (skoreTahu > nejlepsiSkore)
        {
            nejlepsiSkore = skoreTahu;
            nejlepsiTah = vsechnyMozneTahy[i];
        }
        ProvedTahZpet(s, vsechnyMozneTahy[i]);      /* Zahrání tahu zpět */
    }
    return nejlepsiTah; /* Vrátím nejlepší tah */
}
```

Výpis 1: Pseudokód naivního algoritmu

3.3 Minimax

Minimax je jednoduchý algoritmus využívaný ve hrách pro dva hráče jako jsou Piškvorky, Šachy nebo právě Dáma. Slouží k nalezení optimálního tahu. Lze ho použít v případě,

pokud se jedná o logickou hru. To znamená, že hra může být popsaná množinou pravidel a premis. Tento předpoklad je velmi důležitý a bez něj by mohla nastat situace, kdy by algoritmus přesně nevěděl, jaké má možnosti v dané situaci. Algoritmus Minimax se snaží minimalizovat možné ztráty ve hře a maximalizovat zisk (odtud pak název minimax). V Minimaxu rozlišujeme 2 hráče. Hráče MIN a hráče MAX. Z názvu je jasné, že hráč MAX je tím, který se snaží svůj zisk maximalizovat, zatímco hráč MIN se snaží o minimalizování svých ztrát.

Algoritmus byl využit například programem Chinook, o kterém jsme se již zmínili v kapitole 2.3.2. Řekli jsme si také, že Chinook nemůže prohrát. Z toho vyplývá, že algoritmus Minimax má velký potenciál. Aby ho bylo možné používat, předpokládá se existence ohodnocovací funkce, která je schopna určit výhodnosti jakékoliv situace ve hře. [15]

Na Obrázku 6 si vysvětlíme, jak algoritmus funguje. Algoritmus začne z pohledu maximalizujícího hráče. Vytvoří se stromová struktura, kde kořenem stromu bude výchozí pozice hráče MAX tj. uzel A, který se nachází na úrovni 0. Ten má na výběr, jaký tah může zahrát. Tyto tahy jsou reprezentovány uzly na úrovni 1 (B, C a D) a jsou potomky kořenového uzlu stromu. MAX svůj tah odehrál. Hráči se střídají, takže na tahu je MIN. Ten pro každý z možných tahů hráče MAX (pro všechny uzly na úrovni 1) prozkoumá, jaké jsou jeho možnosti táhnout. Opět se vytvoří nové uzly, které tyto tahy reprezentují. Dále pokračuje opět hráč MAX, pak znovu MIN. Tohle přepínání a generování tahů pokračuje až do předem definované hloubky. Když požadované hloubky dosáhneme, nastává ohodnocování stavů, do kterých se hráči zahráním různých tahů dostali. Listy stromu reprezentují všechny možné situace, které mohou nastat po zahrání předem definovaného počtu tahů. Maximální hloubka stromu, která je vstupním parametrem algoritmu nám říká, kolik tahů dopředu má uvažovat. Algoritmus skončil z pohledu hráče MIN, který listy (uzly na úrovni 3) ohodnotil. O úroveň výš hráč MAX vybere největší možnou hodnotu pro každý z uzlů na této úrovni. Hráč MAX si tedy vybírá hodnoty nejvyšší. Naopak v dalším kroku vybírá hráč MIN. Řekli jsme si, že jemu jde o minimalizování svých ztrát, takže se snaží „kazit" hru hráče MAX tím, že ve svém tahu udělá krok, který je pro hráče MAX nejnevýhodnější. Nakonec se hráč MAX dopracuje až na kořenovou úroveň a tím dostane tah, který musí provést, aby dosáhl nejlepšího možného výsledku. Algoritmus je popsán i pseudokódem ve Výpisu 2.

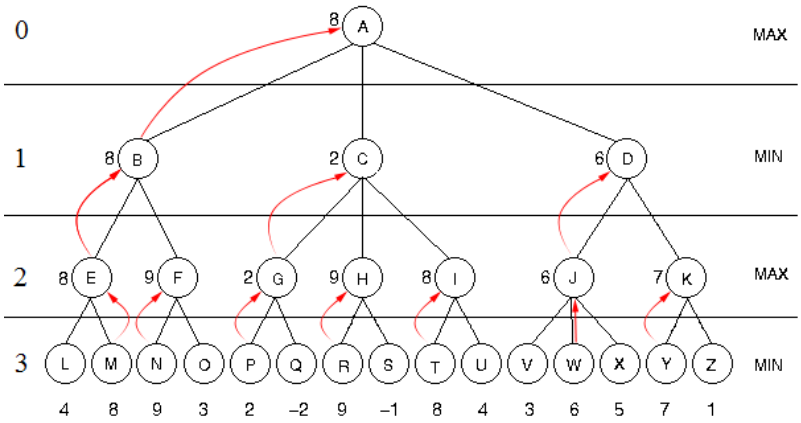
Může nastat i situace, kdy výpočet skončí dřív, než se dosáhne maximální hloubky stromu. Tato situace může nastat, když hráč, který je na tahu už žádné další tahy nemá. V tom případě se vyhodnotí tahy v této hloubce.

Je důležité zdůraznit, že strom, který se tímto způsobem vytváří, je velmi rozsáhlý. Není efektivní ho celý vygenerovat na začátku, uložit do paměti a pak ho procházet. Algoritmus ve stejnou dobu vytváří i prochází strom. Strom se prohledává do hloubky a vždy při návratu o úroveň výš se počítá hodnota uzlu, do kterého se vracíme.

Strom, který algoritmus generuje roste exponenciálně. Velikost stromu závisí na jeho hloubce a větvícího faktoru (označíme jsi ho jako α). Větvící faktor nám udává, kolik potomků bude mít každý z uzlů ve stromu. Pokud bychom měli větvící faktor 3, znamenalo by to, že ve hloubce stromu n bychom nalezli 3^n uzlů. Pokud chceme zjistit úplný počet

Hloubka	Počet uzlů
0	1
1	8
2	64
3	512
...	...
n	8^n

Tabulka 1: Hodnoty pozic hrací desky



Obrázek 6: Stromová struktura algoritmu minimax

vygenerovaných uzlů, musíme udělat součet uzlů na všech úrovních stromu. Počet všech uzlů dostaneme tedy jako: $\sum_{n=0}^{n} \alpha^n$, kde n je hloubka stromu a α je větvící faktor. Průměrný větvící faktor hry Dáma je 8. Z tabulky 1 můžeme vyčíst, kolik uzlů se vygeneruje v jednotlivých hloubkách stromu. Počet všech uzlů při hloubce stromu 6, které se v Dámě vygenerují, si můžeme jednoduše spočítat. Jako faktor větvení použijeme 8. Výpočet bude vypadat následovně:

$$\sum_{n=0}^{6} 8^n = 1 + 8 + 64 + 512 + 4096 + 32768 + 262144 = 299593.$$

Pokud by zpracování jednoho uzlu trvalo 1 milivteřinu, znamenalo by to, že vytvoření stromu by zabralo 5 minut. Snížením maximální hloubky stromu můžeme trvání výpočtu zredukovat.

```
int Max(Stav s, int hloubka)
{
   int nejlepsiSkore = −NEKONECNO;

   VygenerujTahy();
   /* Pokud neexistuje platný tah, nebo jsem dosáhl maximální hloubku, vyhodnotím stav */
   if (vsechnyMozneTahy.pocet == 0 || hloubka <=0)
   {
      return Ohodnot(s);
   }
   for (i = 0; i < vsechnyMozneTahy.pocet; i++)      /* Iterace skrze všechny tahy */
   {
      ProvedTah(s, vsechnyMozneTahy[i]);        /* Zahrání tahu */

      /* Volání do hloubky. (Výpočet z pohledu hráče MIN) */
      skoreTahu = Min{s, hrac.souper, hloubka−1}

      ProvedTahZpet(s, vsechnyMozneTahy[i]);      /* Zahrání tahu zpět */

      /* Je−li tah lepší než dosavadní nejlepší, bude nový nejlepší */
      if (skoreTahu > nejlepsiSkore)
      {
         nejlepsiSkore = skoreTahu;
      }
   }
   return nejlepsiSkore;
}

int Min(Stav s, int hloubka)
{
   int nejlepsi = NEKONECNO;

   VygenerujTahy();
   /* Pokud neexistuje platný tah nebo jsem dosáhl maximální hloubku vyhodnotím stav */
   if (vsechnyMozneTahy.pocet == 0 || hloubka <=0)
   {
      return Ohodnot(s);
   }
   for (i = 0; i < vsechnyMozneTahy.pocet; i++)      /* Iterace skrze všechny tahy */
   {
      ProvedTah(s, vsechnyMozneTahy[i]);        /* Zahrání tahu */

      /* Volání do hloubky. (Výpočet z pohledu hráče MAX) */
      skoreTahu = Min{s, hrac.souper, hloubka−1}

      ProvedTahZpet(s, vsechnyMozneTahy[i]);      /* Zahrání tahu zpět */

      /* Je−li tah lepší než dosavadní nejlepší, bude nový nejlepší */
      if (skoreTahu < nejlepsiSkore)
      {
         nejlepsiSkore = skoreTahu;
      }
   }
   return nejlepsi;
}
```

Výpis 2: Pseudokód algoritmu MiniMax

3.4 Negamax

Negamax je varianta algoritmu Minimax. Liší se jen kratším zápisem. Místo dvou metod MIN a MAX se používá jediná s názvem NEGAMAX. Ta se pak rekurzivně volá vždy s opačným znaménkem pro soupeře. Jinými slovy negujeme vrácenou hodnotu. Odtud pak pochází i samotný název. Algoritmus je popsán pseudokódem ve Výpisu 3. Ke své funkčnosti samozřejmě také vyžaduje existenci ohodnocovací funkce. Ta se ale musí upravit tak, aby byla schopna ohodnotit stav hry z pohledu obou hráčů. Tím, že se znaménko střídá při každém volání funkce, pracujeme s hodnotami následovně. Záporná hodnota je nevýhodná pro hráče na tahu a kladná je naopak pro něj výhodná nezávisle na tom, jestli jde o hráče minimalizujícího nebo maximalizujícího. [15]

```
int Negamax(Stav s, Hrac hrac, int hloubka)
{
   nejlepsiSkore = −NEKONECNO;

   VygenerujTahy();
   /* Pokud neexistuje platný tah nebo jsem dosáhl maximální hloubku, vyhodnotím stav */
   if (vsechnyMozneTahy.pocet == 0 || hloubka <=0)
   {
      return Ohodnot(s);
   }
   for (i = 0; i < vsechnyMozneTahy.pocet; i++)      /* Iterace skrze všechny tahy */
   {
      ProvedTah(s, vsechnyMozneTahy[i]);        /* Zahrání tahu */

      /* Rekurzivní volání do hloubky. (Výpočet z pohledu soupeře) */
      skoreTahu = −Negamax{s, hrac.souper, hloubka−1}

      ProvedTahZpet(s, vsechnyMozneTahy[i]);      /* Zahrání tahu zpět */

      /* Je−li tah lepší než dosavadní nejlepší, bude nový nejlepší */
      if (skoreTahu > nejlepsiSkore)
      {
         nejlepsiSkore = skoreTahu;
      }
   }
   return nejlepsiSkore;
}
```

Výpis 3: Pseudokód algoritmu NegaMax

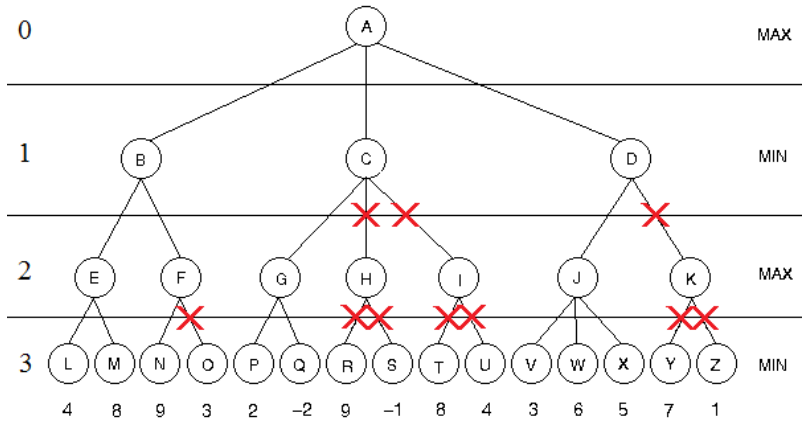
3.5 Alfa-beta ořezávání

Algoritmy Minimax i Negamax jsou výpočetně i časově velice náročné. Existuje způsob, kterým je možné algoritmus zrychlit. Jde o tzv. alfa-beta ořezávání.

Při hraní deskových her už i průměrní hráči dokážou určit, který z tahů nemá smysl zkoumat. Na tomto principu funguje i zmíněná optimalizace. Algoritmus, který chceme optimalizovat, musíme rozšířit o hodnoty alfa a beta. Při prohledávání stromu jsou tyto

Algoritmus	Počet ohodnocených uzlů	Počet oříznutí	Oříznutí v %	Trvání výpočtu
Minimax	1544761	0	0,00%	7,44 s
AlfaBeta	169973	7524	89,1%	0,81 s

Tabulka 2: Optimalizace algoritmu



Obrázek 7: Alfa-beta ořezávání

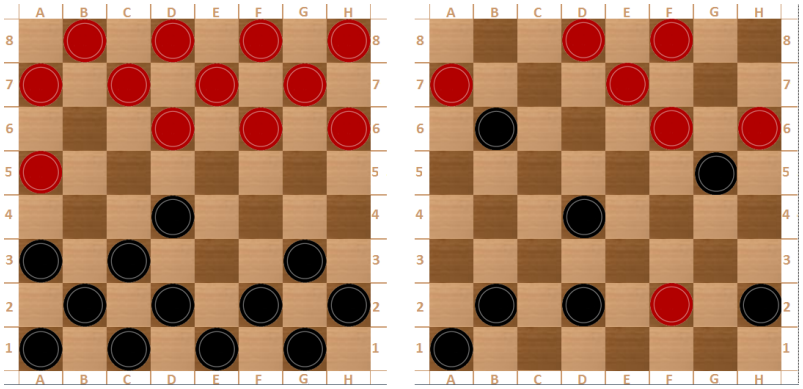
hodnoty aktualizovány a naznačují, jakých výsledků mohou hráči dosáhnout při optimální (bezchybné) hře. Alfa a beta určují jakési meze. Pomocí této meze je možné rozhodnout, které části stromu nebude nutné prohledávat. Tahy, které nemohou ovlivnit konečný výsledek, nebudou brány v potaz.

Mějme situaci, ze které může hráč na tahu provést tahy a, b a c. Tah a bude ohodnocovací funkcí ohodnocen třemi body, tah b jedním bodem a tah c sedmi body. Neoptimalizovaný minimax algoritmus by začal hledat všechny možnosti, které soupeř má pro všechny tři tahy. Optimalizovaný algoritmus si uloží, že tah a byl ohodnocen 3 body. Při zkoumání tahu b zjistí, že se jedná o tah, který je horší (je hůře ohodnocený, než doposud zjištěný nejlepší) a podstromem tohoto tahu se už nebude zabývat. Algoritmus přestane zkoumat reakci soupeře na tah, který zjevně povede k horším výsledkům. Reakce soupeře na tah c se už prozkoumávat bude, protože má hodnotu větší než doposud byla zjištěna.

Jakého zrychlení lze dosáhnout pomocí techniky alfa-beta ořezávání, nelze jednoznačně určit. Závisí na konkrétní situaci, ve které ořezávání aplikujeme. Tabulka 2 porovnává neoptimalizovaný algoritmus Minimax s jeho optimalizovanou verzí (pomocí alfa-beta ořezávání). Hodnoty v tabulce slouží jen pro představu. Ukazují, jakého zlepšení lze optimalizací dosáhnout ve hře Šachy. Pro výpočet jsme použili průměrného větvícího faktoru ($\alpha = 35$). A výpočty probíhaly do hloubky 4. [16]

3.6 Ohodnocovací funkce

Podmínkou využití zmíněných algoritmů je existence tzv. ohodnocovací funkce. Jde o funkci, která vyhodnotí stav partičky. Jelikož ohodnocovací funkci využíváme ve vý-



Obrázek 8: Ekvivalentní situace z hlediska rozdílu počtu figurek

početně složitých algoritmech, měla by být co nejrychlejší a nejjednodušší. V případě deskových her se často jedná o funkci, která pouze spočítá, kolik figurek se nachází na hrací ploše a vrátí rozdíl mezi figurkami hráče a soupeře. Takto napsaná ohodnocovací funkce je rychlá, ale pokud je našim cílem, abychom vytvořili algoritmus, který by měl porážet lidské soupeře, je nutné ji značně rozšířit.

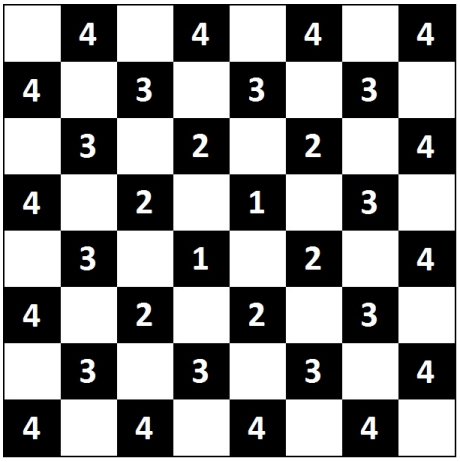
Představme si dvě situace, které znázorňuje Obrázek 8. Na levé hrací desce má každý z hráčů 12 figurek. Rozdíl mezi počtem figurek je tedy nulový (skóre = 0). Na pravé hrací desce mají oba dva hráči po sedmi figurkách. Opět nemá žádný z nich početní převahu a skóre se rovná nule. Algoritmus, který k ohodnocení stavu hry využívá jen rozdílů v počtu, vyhodnotí obě situace jako ekvivalentní. Přičemž je zjevné, že situace na pravé hrací desce je lepší pro červeného hráče. Na pozici $F2$ má figurku, která se v dalším kroku může stát králem. Z pole $A7$ se může dostat přes $C5$ a $E3$ až na pole $C1$. Tím nejen, že získá dalšího krále, ale také způsobí soupeři značné škody a svojí pozicí ho bude opět ohrožovat.

Jakýmsi rozšířením tohoto algoritmu je určení „síly“ hráče nejen podle počtu a typu figurek, ale i podle pozic, které figurky obsazují. K tomu je nutné ohodnotit hrací desku. Jedním z nejčastěji používaných ohodnocení je znázorněné na Obrázku 9.

Problém takto ohodnocené hrací plochy je, že nutí hráče hrát defenzivně. Dáma není hra, kde je efektivní hrát pasivně. Obětování svých figurek k tomu, abychom soupeře dostali do pro něj nevýhodné situace, je důležitým taktickým prvkem.

3.6.1 Ohodnocování pomocí parametrů

Pokud je našim cílem vytvořit umělou inteligenci, která bude výzvou i pro zkušené hráče, měli bychom ohodnocovací funkci rozšířit. Nejvyspělejší algoritmy využívají velké množství různých parametrů, díky kterým je možné lépe určit, jak výhodný je aktuální stav z hlediska některého z hráčů. V předcházejících sekcích byl zmíněn neporazitelný počítačový program Chinook, který využíval k vyhodnocení stavu kromě počtu figurek i různé jiné faktory, jako například počet králů na hrací ploše, počet uvězněných králů, číslo tahu, počet figurek, které mají volnou cestu k tomu, aby se z nich stal král apod. Šachový



Obrázek 9: Ohodnocená hrací deska

program Deep Blue ve své evaluační funkci obsahuje kolem 6 000 různých parametrů, pomocí kterých vyhodnocuje stav. [2] [3]

Jak už bylo zmíněno, cílem této diplomové práce není vytvoření neporazitelné umělé inteligence, proto jsem zvolil jednodušší ohodnocovací funkci. Všechny z výše popsaných algoritmů počítají skóre pomocí té samé ohodnocovací funkce. Často se také parametrům přiřazují váhy. Výsledné skóre pak vzniká jako suma všech vlastností vynásobená jejich váhou, tedy podle vzorce:

$$skore = (vaha_1 * parametr_1) + (vaha_2 * parametr_2) + \ldots + (vaha_n * parametr_n).$$

4 Evoluční výpočetní techniky

Definice 4.1 *Biologická evoluce se chápe jako změna v genetickém materiálu populace v čase. [17]*

Z definice 4.1 vyplývá, že biologická evoluce je proces, který postupem času přináší jisté změny. Důsledkem těchto změn je, že se život na Zemi neustále vyvíjí. Biologické modifikace, ke kterým dochází, zpravidla přinášejí vstupní entity evoluce nějaké zlepšení. Důkazem tohoto tvrzení může být například změna zabarvení tělního pokryvu některých živočichů v závislosti na prostředí, ve kterém se dlouhodobě zdržují.

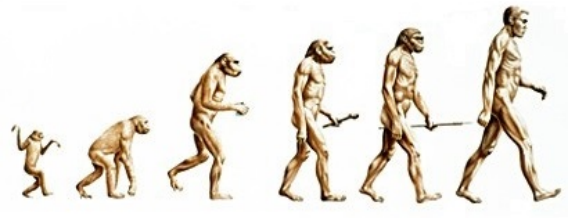
Příkladem by mohla být liška polární. Proces evoluce zajistil, že se tento živočich svým zabarvením přizpůsobil svému prostředí. Zvíře tak splývá se svým okolím, čímž je jí umožněno nepozorovaně se přiblížit ke své kořisti. Můžeme tedy říci, že evoluce postupem času lišku „vylepšila“.

Dalším dobrým příkladem je sám člověk. Podle Darwinovy evoluční teorie mají lidé a opice stejného předka, ze kterého se vyvinuli. Mylná je interpretace Darwinovy teorie, podle které se lidé vyvinuli z opice. Oba primáti se přizpůsobili svému životnímu prostředí jiným způsobem. Přirozeným prostředím opic jsou stromy. Jejich končetiny mají dlouhé a zahnuté prsty, které jim umožňují lepší uchopení větví. Přirozeným životním prostředím člověka se stala zem. Prsty člověka jsou rovné a kratší. Palec na noze se zvětšil, abychom dokázali udržet rovnováhu při chůzi na dvou nohách. Nejznámější ilustraci evoluce od amerického umělce Rudolpha Zallingera můžeme vidět na Obrázku 10.

Podobných případů, kde se živočichové nebo rostliny genetickými změnami přizpůsobují svému prostředí a okolnostem, nalezneme v přírodě nespočetně mnoho. Proto je biologický mechanismus evoluce vnímán vědci často jako jistý druh „optimalizace“, kterou příroda provádí. Právě tento jev inspiroval mnoho z nich, aby se začali zabývat myšlenkou aplikace principů evoluce ve výpočetní technice. Výsledkem tohoto bádání jsou tzv. evoluční výpočetní techniky. [17]

4.1 Historie evolučního programování

Myšlenka využití principů Darwinovy evoluční teorie pro automatizované řešení úkolů vznikla v padesátých letech minulého století. Biologové se pomocí počítačů snažili namodelovat aspekty přirozené evoluce. V šedesátých letech téhož století na tuto práci navázalo hned několik skupin vědců. Všichni se snažili využít stejných principů evoluce v různých oblastech informatiky. Ve Spojených státech Lawrence J. Fogel A.J. Owens, a M.J Walsh [18] představili svoji myšlenku evolučního programování. Jejich krajan John H. Holland říkal své metodě genetický algoritmus. V Německu se tématem zabývali vědci Ingo Rechenberg a Hans-Paul Schwefel, G.J. Friedman, H.J. Bremermann a mnoho dalších. [19], [20] Ti představili světu své tzv. evoluční strategie. Právě němečtí vědci se snažili aplikovat tyto evolucí inspirované algoritmy na poli umělé inteligence. Jejich práce však nevzbudila velkou pozornost. Výsledný algoritmus se hodně podobal tzv. gradientnímu algoritmu. Jde o iterační algoritmus, který začíná s „náhodným“ řešením problému a snaží se dopracovat k lepšímu řešení. Řešení hledá tak, že postupně mění jeden z parametrů, které



Obrázek 10: Ilustrace evoluce člověka

řešení ovlivňují. Pokud změna vytváří lepší řešení, stane se základem nové iterace. Algoritmus běží až do doby, kdy už není možné nalézt žádné další zlepšení. Principem evoluční strategie bylo vytvořit potomka, který byl následně změněn. Pokud byl potomek lepší než jeho předchůdce, byl v další iteraci použit. Pokud byl horší, zahodil se a vytvořil se nový potomek. Takový přístup se z části podobá genetickému algoritmu, o kterém si více řekneme později. Křížení a mutace zde ještě ale nebyly definovány.

Úspěchů dosáhli i američtí vědci. Lawrence J. Fogel aplikoval evoluční programování k nalezení řešení problému. Princip algoritmu byl podobný evoluční strategii. Jednotlivá možná řešení byla reprezentována konečným automatem. Jeden z nich se náhodně zmutoval (pozměnil), porovnaly se výsledky a dále se pracovalo s nejlepším z nich. Mutace jako taková je zde už přítomná, křížení zatím ne.

John Holland [21] přišel v roce 1962 s myšlenkou simulovat biologickou evoluci nejen pomocí mutace, ale také pomocí přirozeného výběru a křížení. Vytvořil tak podobu algoritmu, která se používá dodnes. Mezi evoluční procesy, které dnes EA využívají, patří: dědičnost, mutace, přirozený výběr a křížení.

4.2 Využití

Už v 80. letech se evoluční programování využívalo prakticky ve všech možných oborech k řešení široké škály problému (obarvení grafu, rozeznávání vzorů, strukturální optimalizace, atd.). Tyto algoritmy se využívají dodnes. Nárůst výpočetní síly počítačů umožnil aplikování EA i v komerčním sektoru. [22]

4.2.1 Hry

K. Chellapilla a J. Fogel [23], jsou autoři jedné z nejpřesvědčivějších demonstrací síly evolučního programování. Povedlo se jim pomocí tzv. genetického algoritmu (GA) vytvořit neuronovou síť, která byla schopná hrát Dámu. Největším problémem podle autorů bylo vymyslet funkci, která dokáže ohodnotit, jak dobře počítač hrál. V Dámě nalezneme mnoho měřítek pro srovnávání toho, jak dobře nebo špatně někdo hraje. Není efektivní brát v potaz jenom pouhý poměr výher, proher a remíz. Přesto se autorovi povedlo vytvořit algoritmus, který umí porážet své lidské soupeře tak, že k ohodnocení využívá jen počet figurek na hrací ploše a jejich pozici.

Hodnota	Popis
0	neobsazená pozice
-1	pozice obsazená figurkou patřící soupeři
1	pozice obsazená figurkou patřící hráči
$-x$	pozice obsazena soupeřovým králem[1]
$+x$	pozice obsazena králem hráče na tahu[1]

Tabulka 3: Hodnota prvků seznamu podle obsazenosti příslušné pozice

Hrací deska byla reprezentována seznamem (datová struktura známá z informatiky, kde jednotlivé prvky jdou „za sebou“). Každá z „dosažitelných“ pozic hrací desky 8x8 políček byla reprezentována jako prvek v seznamu. Pozic na takové ploše je 64, ale na rozdíl od Šachů se mohou figurky pohybovat jen po jedné z barev hrací desky. Výsledná délka seznamu je tedy poloviční, 32. Prvky seznamu uchovávaly hodnotu v závislosti od toho, jaká figurka se na pozici nacházela 3.

Algoritmus vytvořil stromovou strukturu, kde uzly reprezentovaly jednotlivé situace, které by mohly nastat. Strom byl hloubky 4, tj. uvažovalo se 4 kroky dopředu. Jednotlivé situace byly ohodnoceny hodnotou z intervalu od -1 do 1 v závislosti na tom, jak dobrá nebo špatná je situace pro hráče na tahu.

Algoritmus začal s populací patnácti náhodně vygenerovanými jedinci. Pro každého jedince se vytvořil potomek, který byl mírně pozměněn. Celková populace třiceti hráčů pak hrála proti sobě. Každý z hráčů 5 her s náhodně zvoleným soupeřem. Výhra byla ohodnocena jedním bodem, prohra odečtením dvou bodů. Hráči, kteří dosáhli nejlepšího skóre, pak vytvořili novou generaci. Chellapilla a Vogel vytvořili přes 800 generací, což jim v té době zabralo více než půl roku. Výsledkem byla UI, která v turnaji proti lidským spoluhráčům dokázala vyhrávat a porážet i nejzkušenější. Uvádí se, že algoritmus porazil přes 90% všech svých soupeřů.

Pozdější verze algoritmu, která byla pojmenována jako Anakonda, dosahovala tak dobrých výsledků, že se autoři rozhodli postavit ji proti neporazitelnému Chinook algoritmu. Chinook byl konfigurován tak, aby „myslel“ 5 tahů dopředu. V případě Anakondy to bylo 8 tahů. Porovnávat algoritmy se stejnou hloubkou by nemělo smysl, Chinook už dříve dokázal, že v případě „férové“ hry je neporazitelný. Anakondě se z deseti her povedlo vyhrát dvakrát a další 4 skončily remízou. Turnaj sice vyhrál Chinook, ale autoři dokázali, že jejich revoluční UI je přinejmenším konkurenceschopný.

4.2.1.1 Matematika: R. Haupt a S. E. Haupt [22] pomocí GA vytvořili program, který dokáže vyřešit nelineární parciální derivaci vysokého řadu. Dvě možná nejčastěji používané matematické nástroje vědců jsou právě obyčejné a parciální diferenciální rovnice. Obecně o nich neuvažujeme jako o minimalizačních problémech. Pokud bychom ale chtěli nalézt hodnoty, ve kterých je diferenciální rovnice rovná nule, můžeme hledat minimum absolutní hodnoty rovnice. Tímto způsobem Haupt ukázal, jak je možné využít GA k řešení nelineárních diferenciálních funkcí, které jsou formálně neintegrovatelná.

[1] Hodnota x nebyla autorem specifikována.

4.2.1.2 Chemie: Pomocí výkonného laseru je možné rozložit komplexní molekuly na jednodušší. Laser vyšle krátké elektromagnetické impulzy, které molekuly rozštěpí. Jednotlivé impulzy trvají jen několik femtovteřin (10^{-15} vteřin). Tento proces se často využívá jak v chemii, tak i v mikroelektronice. Fázovou modulací laserového impulzu je možné tuto reakci kontrolovat. V případě velkých molekul je ale velice náročné určit tvar impulzu, který by vedl k požadovanému výsledku.

Assion a kol. [26] tento problém vyřešili použitím EA. Jejich řešení nespočívá v zadávání přesných hodnot, které by vedly k optimálnímu řešení. Autoři se pomocí EA snaží k výsledku dopracovat postupně. Nejdřív se vyšle impulz, který molekuly rozštěpí a zkontroluje se výsledek. Pokud výsledek není požadovaný, parametry impulzu se mírně změní a proces se opakuje. Autoři tvrdí, že jejich algoritmus automaticky nalezne nejlepší možnou konfiguraci laseru nezávisle na složitosti zkoumané molekuly.

4.2.1.3 Elektrotechnika: Programovatelná hradlová pole jsou speciální číslicové integrované obvody obsahující různě složité programovatelné bloky, které jsou propojené konfigurovatelnou maticí spojů. [25] Obvod je možné řídit softwarem, který si do něj můžeme nahrát. V závislosti na instalovaném softwaru je možné využít hradla pro řízení široké palety hardwaru. Právě díky své programovatelnosti, snadnému návrhu, flexibilitě, neustále klesajícím cenám a zvolna se snižující spotřebě energie vlastním čipem, nacházejí dnes tyto obvody uplatnění v široké škále aplikací.

Dr. Adrian Thompson vytvořil pomocí programovatelného hradlového pole a evolučních principů zařízení, které umí rozpoznávat lidskou řeč. Populaci algoritmu tvoří náhodně vygenerované řetězce, které slouží jako konfigurace hradel. Algoritmus tedy zkoušel různé konfigurace, které by mohly vést k rozpoznávání klíčových slov. Thompson svého cíle dosáhl po vytvoření přibližně 3 000 generací. Výsledné zařízení umí nejen rozpoznávat klíčová slova, ale také rozlišovat je na různých frekvencích. Jak přesně zařízení funguje, se dodnes neví. Autor nikdy nepřišel na to, jak přesně zařízení rozpoznává řeč. Největší záhadou je skutečnost, že po důkladném prozkoumání se přišlo na to, že některá z hradel vůbec nebyla nijak propojena s ostatními, přesto když se přestala napájet, zařízení přestalo fungovat. [24]

4.2.1.4 Vojenství: Kewley a Embrechts [27] využili GA k nalezení optimálního taktického plánu pro vojenské bitvy. Vytvoření plánu pro bitvu je náročný úkol, který často tíží mnohé zkušené vojenské profesionály. Ti často musí rozhodovat pod velikým tlakem, co může mít za důsledek prohlédnutí nějakého důležitého faktoru, který může významně ovlivnit výsledek bitvy. I jednoduchá taktika vyžaduje prozkoumání velkého počtu různých faktorů, jako například: minimalizování ztrát, způsobení největší škody nepříteli, ovládnutí jistého území, šetření se zdroji atd. Pro lidi to může často znamenat neřešitelný problém. Kewley a Embrechts se pokusili tento proces automatizovat.

Autoři vytvořili počítačový program pro simulování bitev. Vstupem programu je požadovaný výsledek bitvy. Program se pak pokouší nalézt optimální řešení situace, přičemž bere v potaz dostupné zdroje, podobu terénu, rychlost, přesnost jednotek a mnoho dalších faktorů. Správnost výsledku byla ověřena pomocí lidských expertů. Program i experti se

snažili vyhodnotit stejnou situaci a nalézt nejlepší možné řešení. Bitvy byly simulovány podle všech výsledných řešení. Postup, který definoval počítač, se ukázal jako nejlepší. Stratégové dostali možnost podívat se na tento plán a dle libosti upravit části, o kterých si myslí, že by mohly být provedeny lépe. Další simulací se dokázalo, že nejlepší řešení je původní, nekorigované řešení nabízené počítačem.

4.2.1.5 Rozpoznávání vzorů Rizki, Zmuda a Tamburino [28] vytvořili pomocí EA systém pro rozeznávání vzorů. Aktuální trend je takový, že problematiku rozeznávání vzoru stále častěji řeší algoritmy, které jsou schopné učit a rozvíjet se. Pouhé předdefinování vzoru se jeví jako neefektivní. Naopak definovat vzory a nechat, ať si z nich algoritmus vybere, vypadá jako správné řešení.

Autoři podrobili svůj algoritmus následujícímu testu: program dostal jako vstup signál. Tento signál byl radarovým odrazem různých letounů. Odrazový signál stejného letounu se může značně lišit v závislosti na různých faktorech, jako například: úhel v jakém se odráží nebo nadmořská výška, ve které je letoun nasnímán. Na druhou stranu, signály z různých letadel si mohou být hodně podobné. Úkolem algoritmu bylo rozeznat jednotlivé letouny pouze na základě jejich odrazového signálu. Algoritmus dokázal správně rozeznat až na pár výjimek všechny letadla. Procentuální úspěšnost algoritmu se pohybovala kolem 97%, čímž předčila všechny doposud známé techniky rozeznávání vzoru.

4.3 Evoluční algoritmy

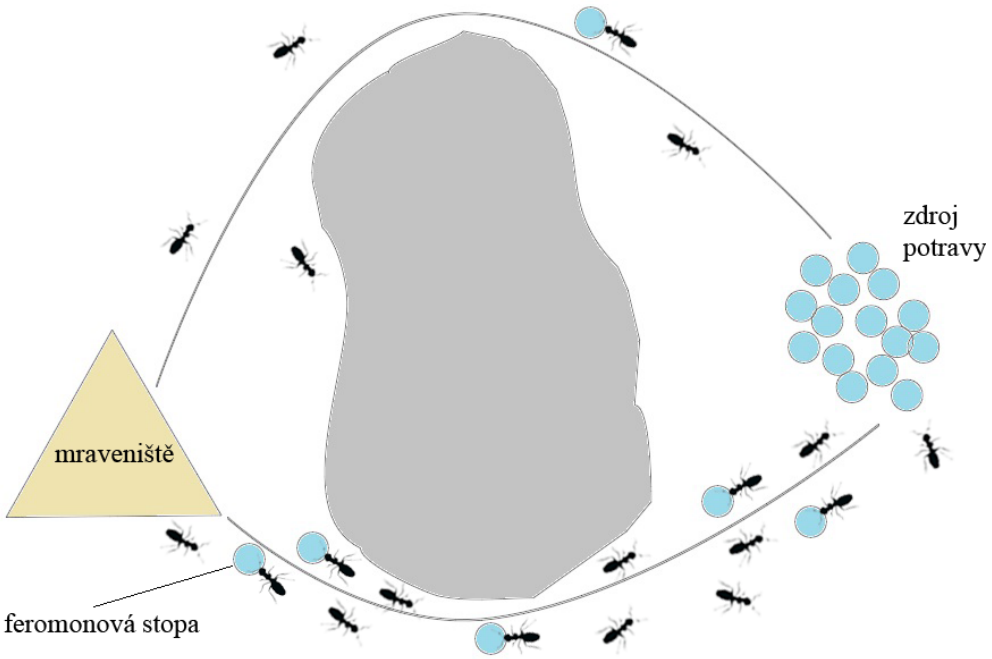
Do skupiny EA řadíme algoritmy, které byly inspirovány jevy pozorovatelnými v přírodě. Mezi nejznámější EA patří například: optimalizace mravenčích kolonií, optimalizace včelím rojem, SOMA, nebo GA.

V další části si stručně představíme některé z nich. Rozsáhlejší kapitolu budeme věnovat GA, který je z hlediska této diplomové práce důležitý.

4.3.1 Optimalizace mravenčí kolonií

Optimalizace mravenčích kolonií (Ant Colony Optimization, ACO) je algoritmus inspirovaný komplexním společenským chováním mravenců. Zejména jejich schopnost nalézt nejkratší možnou cestu k potravě je pozoruhodná. Právě tento proces se stal základní myšlenkou třídy tzv. „mravenčích algoritmů“.

Proces hledání potravy probíhá u mravenců následovně (obrázek 11). Z mraveniště vyrazí několik mravenců všemi směry. Pokud některý z nich narazí na překážku, náhodně zvolí směr, kterým překážku obejde. Ve chvíli, kdy narazí na zdroj potravy, vyšle signál ostatním. Mravenci mezi sebou komunikují pozměněním svého okolí tak, že značkují cestu pomocí feromonu. Mravenec s potravou se vrátí do mraveniště a cestu k potravě označí. Pro zbytek mraveniště to bude znamenat, že při rozcestí upřednostní tuto cestu a nebude si volit směr náhodně. Pokaždé, když některý z mravenců nalezne na konci cesty potravu, cestu dodatečně označkuje a feromonová stopa bude silnější. Feromonová stopa postupně slábne a pokud je cesta dlouhá, slábne rychleji. V případě kratší cesty se po trase



Obrázek 11: Ilustrace hledání potravy mravenci

za stejnou dobu otočí více mravenců, což znamená, že bude tato trasa označena silnější stopou. Tímto způsobem pak jedinci upřednostní kratší cestu. [31]

Na tomto principu funguje algoritmus ACO, kterým lze efektivně řešit problémy typu hledání nejkratší cesty v grafu. Využívají se zde váhy k simulaci feromonové stopy. Tyto váhy, stejně jako značkování mravenců jsou silnější, pokud se cesta jeví jako optimální.

Slábnutí feromonu je také imitováno, což zamezuje algoritmu zaseknout se při nalezení lokálního extrému. ACO byl úspěšně aplikován například na problém obchodního cestujícího.[31]

4.3.2 Samoorganizující se migrační algoritmus

Samoorganizující se migrační algoritmus (Self-Organising Migrating Algorithm, SOMA) je algoritmus založený na *„samoorganizujícím chování skupin osob v sociálním prostředí“* [33]. I navzdory tomu, že není založen na principu biologického rozmnožování (nedefinuje žádné individua jako rodiče a nevytváří potomky), můžeme algoritmus zařadit do skupiny EA. Stejně jako všechny algoritmy z této skupiny, i SOMA pracuje s populací potencionálních řešení. Místo vytváření nových generací, SOMA pracuje s jednou jedinou a žádné další řešení se negenerují. Jednotlivá řešení jsou reprezentována jako vektory. Jedinci spolupracují na řešení konkrétní problematiky. Pohybují se v prostoru možných řešení. V průběhu tzv. migračních kol mění svoji polohu. [33]

4.3.3 Optimalizace rojením částic

Optimalizace rojení částic (Particle Swarm Optimization, PSO) [32] je optimalizační metaheuristická technika představená v roce 1995. PSO je inspirován chováním hejna ptáků při hledání potravy. Populaci zde nazýváme hejnem a tvoří ji tzv. částice. Částice jsou náhodně rozmístěny v prostoru a pohybují se v něm. Jednotlivé částice tak prohledávají prostor možných řešení, přičemž se snaží následovat nejlepší z nich. Jednotlivé části mezi sebou komunikují a navzájem se ovlivňují. Také si pamatují svoji nejlepší polohu a nejlepší polohu okolních částic. Výpočet probíhá v iteracích. V každé iteraci je pohyb částice ovlivněn vlastní nejlepší pozicí, ale také se snaží přiblížit k nejlepší pozici, kterou nalezla některá ze zbytku populace.

4.3.4 Diferenciální evoluce

Diferenciální evoluce (Differential evolution, DE) je další z optimalizačních technik, které iterativně modifikují populaci možných řešení, za účelem nalézt optimální řešení. Zjednodušený princip fungování je následující. Nejdříve se náhodně vygeneruje počáteční generace jedinců, poté se ve všech iteracích pro každého jedince x vykonají následující kroky:

1. Vytvoříme „šumový vektor“: $v = a + (b - c)$, kde a, b a c jsou tři rozdílní jedinci z populace, které jsme si náhodně vybrali.
2. Vektory x a v si mezi sebou náhodně vymění vektorové složky a vytvoří se zkušební vektor v'. Musí proběhnout alespoň jedna výměna.
3. Pokud zkušební vektor bude lepší než jedinec x, bude x nahrazen tímto vektorem. Pokud nebude lepší, v' se zahodí.

Tímto způsobem se algoritmus snaží konvergovat k nejlepšímu řešení. DE také využívá procesy křížení a mutace jako například GA. [34]

5 Genetický algoritmus

GA je programovací technika imitující biologickou evoluci, využívaná k nalezení řešení složitých problémů. Vstupem takového algoritmu jsou potencionální řešení dané problematiky. Jednotlivá řešení jsou ohodnocena tzv. účelovou funkcí.

Účelová funkce rozhoduje o tom, jak blízko se nachází jedinec k dosažení stanovených cílů. Funkce přidělí jedincovu hodnotu zvanou fitness, která tuto vzdálenost definuje.

Jednotliví kandidáti mohou být známým řešením problematiky. Od GA se pak očekává, že z těchto řešení vybere nejlepší a pokusí se je optimalizovat. Dalším přístupem je vygenerovat řešení náhodně.

Pokud jsou řešení generována náhodně, je velká pravděpodobnost, že některá z nich budou prakticky nepoužitelná a danou problematiku „nevyřeší“. Tyto možnosti budou samozřejmě vyřazeny. Některé z náhodně generovaných řešení mohou být funkční a mohou vykazovat známky správného řešení.

Tito kandidáti jsou ponecháni a jsou nuceni k reprodukci. Vytváří se z nich několik kopií. Tyto kopie ale nejsou perfektní. Jednotlivé části jsou mírně pozměněny. Takhle vytvořená nová populace možných řešení pak formuje novou generaci. Fitness nové generace bude opět vyhodnocen. Může se stát, že někteří jedinci se stali horšími, nebo se pouze nezlepšili. Tito jedinci budou opět odstraněni. Můžeme zde ale nalézt jedince, kteří se naopak zlepšili a jejich řešení je blíž k optimálnímu. Opět si je zvolíme jako základ nové generace a donutíme je k vytvoření potomků, které náhodně změníme. Tento proces se opakuje mnohokrát. Očekáváním je, že fitness populace se zvýší každou generací. Pokud takových generací vytvoříme stovky nebo dokonce tisíce, můžeme se dopracovat ke kvalitnímu řešení problému. [34]

GA už mnohokrát prokázal, že se jedná o velice silný nástroj pro řešení problému. Řešení, ke kterým se pomocí GA dopracujeme, jsou často efektivnější, všestrannější a více sofistikovanější jako cokoliv, co lidští inženýři produkují. Nejednou se už stalo, že výsledek, ke kterému se algoritmus dopracoval, překvapil i samotné autory (viz [28] nebo [27]).

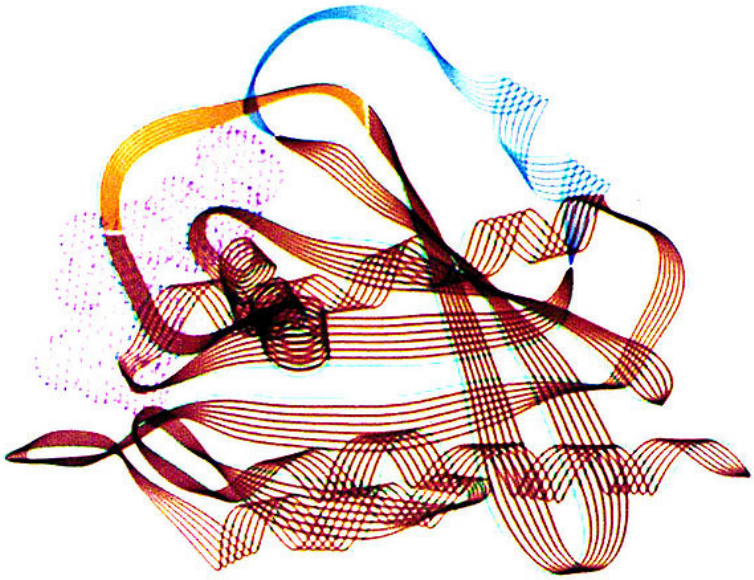
5.1 Populace

Jak jsme si už říkali, vstupem GA je množina možných řešení problému. Této množině říkáme populace a prvky populace nazýváme jedinci. K tomu, aby bylo možné GA použít, musíme nejdříve algoritmu tyto vstupy předat v takové podobě, které bude rozumět. Jedním z často využívaných způsobů je jednotlivé prvky populace reprezentovat řetězcem obsahujícím pouze nuly a jedničky (Obrázek 12). Vstup v takové podobě je pro počítač přirozený a dokáže ho lehce zpracovat. Jednotlivá čísla a jejich umístění představují jistou hodnotu spojenou s řešením. Dalším často používaným postupem je reprezentovat jedince jako pole celých čísel. Používání celých čísel místo jedniček a nul vnáší do řešení větší sofistikovanost a je jednodušší se dopracovat k řešení. [29]

Tato technika byla využívána i v příkladu, který popsala autorka Melanie Mitchellová [29]. V tomto příkladu se GA využíval k nalezení třírozměrné struktury bílkovin (Obrázek 13) na základě posloupnosti aminokyselin. Aminokyseliny jsou základním stavebním

1	0	1	1	0	0	0	1

Obrázek 12: Binární kódování jedince



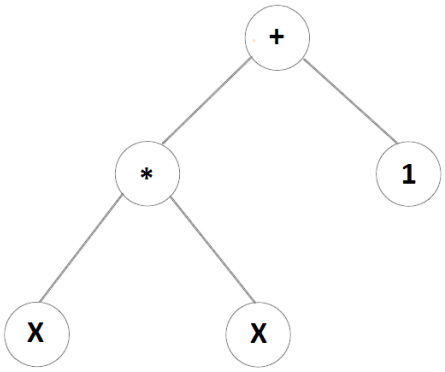
Obrázek 13: Třírozměrné zobrazení struktury bílkovin [30]

kamenem bílkovin. Bílkoviny se skládají spojením jednotlivých aminokyselin. Ve chvíli, kdy se všechny aminokyseliny spojí, vytvoří se třírozměrná struktura, jejíž tvar závisí na tom, které z aminokyselin se přitahují a které se naopak odpuzují. „Tvar“ bílkovin určuje, zda-li se jedná o stavební, transportní, katalické, signální nebo jiné bílkoviny.) Celými čísly zde byly reprezentovány tzv. „dihedrální úhly“ (úhel umožňující popsat jednoduché vazby mezi dvěma atomy jakožto úhel svíraný rovinami, v nichž leží). Takový úhel může nabývat hodnot z intervalu $(+180)$ až (-180) stupňů, takže celočíselná reprezentace byla zde nezbytná.

Třetím způsobem jak reprezentovat jedince z celkové populace je použít řetězce se znaky (písmeny) místo čísel. I tento příklad nalezneme v publikaci [29]. GA bylo zde využito k vyvinutí sady pravidel, tzv. bezkontextové gramatiky. Tato bezkontextová gramatika byla později využita k vytvoření neuronové sítě sloužící k řešení mnoha problémů. V bezkontextových gramatikách využíváme různých matematických znaku $(+, -, *, (,))$, písmen i číslic, proto by bylo nedostačující použít celočíselné pole.

Způsoby, které jsme si popsali, uchovávaly informace buď v řetězci nebo v poli. Možné je také využití stromové struktury k ukládání jednotlivých jedinců (Obrázek 14).

Všechny tyto popsané způsoby mají jednu společnou vlastnost. Změny, které nad nimi musíme provádět jdou jednoduše aplikovat. V případě řetězce s jedničkami a nulami stačí čísla zaměnit. V případě celočíselného pole můžeme k hodnotě přičíst nebo odečíst



Obrázek 14: Stromová struktura výrazu: $x^2 + 1$

libovolnou hodnotu. U pole naplněného znaky stačí znak vyměnit za jiný. Ve stromové struktuře můžeme změnit hodnotu uzlu nebo nahradit podstrom jiným podstromem.

5.2 Výběr

Jak jsme si popisovali v úvodu této sekce, GA vytváří v každé iteraci novou generaci populace. Z té se vybere část, která se rozmnoží a vytvoří jedince, kteří budou reprezentovat další generaci. Jde o uplatnění Darwinovy teorie, podle které nejlepší jedinci přežijí a vytvoří potomky. O tom, kteří jedinci budou vybráni, rozhoduje jejich fitness. Fitness jim přiřazuje účelová funkce(viz refsec:GA). Existuje několik způsobů jak rozhodnout, která část populace je vhodná k reprodukci a „přežití“. Fitness funkce může hodnotit celou populaci a vybrat z ní nejlepší jedince, nebo je hodnotit po skupinách a vybrat nejlepší z nich. Některé z následujících možností, jak vybrat nejvhodnější jedince, které si zde popíšeme, je možné využít současně. Jejich kombinací dosáhneme vyšší preciznosti při selekci. [29]

5.2.1 Ruletový výber

Nejlepší jedinci mají větší šanci, že budou vybráni než ti, kteří si podle fitness funkce vedou hůř. Tento způsob výběru využívá faktoru náhody. I jedinci, kteří mají nejmenší fitness, mají šanci postoupit do další generace. Jejich šance je však malá, zatímco pravděpodobnost, že postoupí silnější jedinec je větší. Výběr si můžeme představit jako „nespravedlivou“ ruletu, kde každý z jedinců dostane větší nebo menší část rulety. Ruletou se zatočí a u kterého jedince se zastaví, ten bude vybrán. [34]

5.2.2 Rank selekce

Je podobný ruletovému výběru. Jedinci jsou oznámkováni celočíselnou hodnotou jako ve škole. Nejlepší z jedinců dostane číslo 1, druhý v pořadí číslo 2, atd. Říkáme, že jim přiřazujeme hodnost. Na základě této hodnosti pak dostanou část z rulety. Opět dostává lepší z jedinců větší část rulety, ale rozdíly nejsou tak veliké jako u ruletového výběru.

Pokud by byl rozdíl mezi prvním a druhým nejlepším jedincem příliš velký, znamenalo by to, že zbytek populace má mizivou šanci postoupit do další generace. V případě výběru založeného na hodnostech jsou rozdíly zmírněny. [34]

5.2.3 Elitismus

Princip elitismu spočívá ve výběru jedince, který byl účelovou funkcí ohodnocen jako nejvhodnější. Většinou se v GA do další generace nedostává jen jeden nejlepší jedinec, ale hned několik z nich, kteří si vedli nejlépe. Někdy se také do další generace přidávají i jedinci, kteří byli nejlepší v předcházejících generacích a to pro případ, že nová generace nepřinese žádné lepší. Elitismus se často doplňuje ruletovým výběrem. [34]

5.2.4 Generační výběr

Potomci současné generace vytvoří novou generaci. Žádný jedinec z předchozí generace nebude zachován. [24]

5.2.5 Kombinovaný výběr

Vytvoří se několik potomků, ti se přidají do současné generace, kde nahradí některé ze slabších jedinců a pak se ze současné generace stane nová generace. [24]

5.2.6 Hierarchický výběr

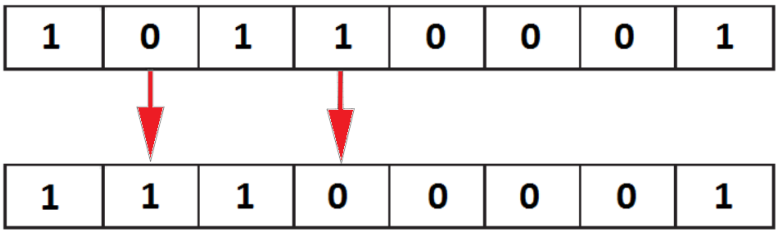
Výběr probíhá ve více kolech. Nejdřív se provede „hrubý“ výběr jedinců, podle méně diskriminujících pravidel a v dalších iteracích se volí přísnější pravidla. Tento způsob výběru je vhodný pro velké populace. Prvotní výběr, který je rychlý, porovnává velké množství jedinců, zatímco preciznější a pomalejší výběr porovnává už jen malý fragment populace. [24]

5.2.7 Turnaj

Jedinci mezi sebou soupeří. Vítěz turnaje bude vybrán pro novou generaci. Turnaj může fungovat na principu round robin (každý z každým), nebo je možné celkovou populaci rozdělit na menší skupiny. Jedinci soupeří mezi sebou v rámci těchto skupin a pak se vybere vítěz z každé z nich. [24]

5.3 Rozmnožování

Ve chvíli, kdy se nám povedlo vybrat vhodného jedince je nutné je mírně pozměnit, aby se vytvořili noví jedinci. Od změn, které provádíme očekáváme, že povedou k lepším výsledkům. Existují dva základní způsoby, jak lze tyto modifikace provádět. Mutace a křížení.



Obrázek 15: Mutace

5.3.1 Mutace

Jednodušší z modifikací je tzv. mutace. Stejně jako u biologické mutace dochází ke změně genetické informace, proto i zde nastávají drobné změny v některých částech kódu. Mutace může pozměnit jeden, dva nebo více částí původního kódu. Tuto hodnotu je důležité zvolit optimálně, stejně jako pravděpodobnost, s jakou mutace nastane. Ne vždy musí být přínosné, když se změní více informací. Vysoká pravděpodobnost mutace může mít za důsledek to, že bude algoritmus prohledávat příliš široké spektrum jedinců a nesoustředí se na ty, kteří se jeví jako nejvhodnější. Na straně druhé, pokud bychom se pokusili mutaci vynechat, algoritmus by našel nejlepšího jedince současné generace a k lepším výsledkům by se už nemusel dopracovat.

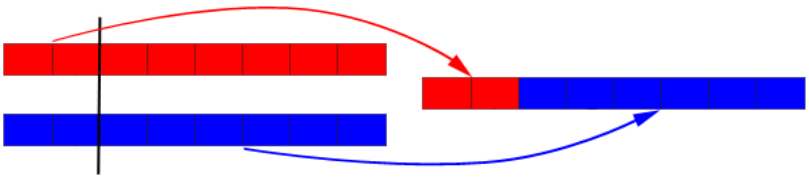
Proces mutace je znázorněn na Obrázku 15. Na horní části obrázku nalezneme původního nezmutovaného jedince a na spodní části obrázku jedince po mutaci. Změnili se zde dvě hodnoty. Na druhé pozici se nám změnila 1 na 0 a na třetí pozici proběhla změna v opačném směru. Tento příklad ilustruje změnu dvou prvků, ve skutečnosti jich může být libovolný počet. Je také možné tento počet předem definovat. Prvky se vybírají náhodně.

V případě celých čísel bychom mohli přičíst, resp. odečíst konstantní hodnotu nebo hodnotu změnit o několik procent. Pokud bychom místo čísel pracovali se znaky, stačí zaměnit jeden znak za jiný. [34]

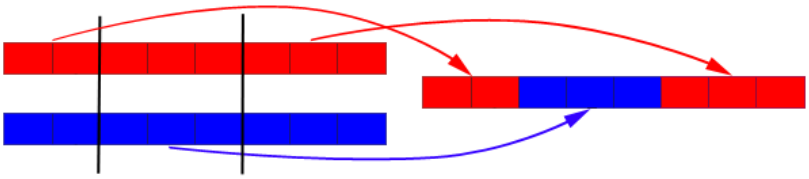
5.3.2 Křížení

Druhá z metod, která slouží k modifikací jedinců je tzv. křížení. Křížení je proces, při kterém se vyberou dva jedinci (říkáme jim rodiče), kteří si vymění části svých kódů, čímž vytvoří nového jedince. Této nově vytvořené entitě říkáme potomek. Potomek je kombinace svých rodičů. Tento proces napodobuje analogický proces rekombinace chromozómů při pohlavním rozmnožování, známý z biologického světa.

Často se využívá tzv. jednobodové křížení (obrázek 16), kde se náhodně určí bod z obou rodičů. Tento bod je hranice určující, které části kódu budou změněny. Potomek obdrží od jednoho z rodičů část svých informací od začátku až po tento bod. Od druhého rodiče obdrží část od zvoleného bodu až po konec. Prakticky se tak informace rodičů rozdělí na dvě části, které nemusí být stejné velikosti. Potomek pak uchovává v sobě jednu část od každého z rodičů.



Obrázek 16: Jednobodové křížení



Obrázek 17: Vícebodové křížení

Další formou křížení je vícebodové křížení (Obrázek 17), zvané také uniformní. Jedná se zde o stejný princip jako u jednobodového křížení s tím rozdílem, že se nerozdělují rodiče na základě jednoho bodu, ale na základě více bodů. Potomek pak například může obdržet začátek a konec kódu od jednoho rodiče a střed kódu od druhého rodiče. [34]

Stejně jako je to v biologickém světě, ani tady nemáme zaručeno, že vzniklý potomek bude lepší nebo alespoň stejně dobrý, jako jeho rodiče. Může se stát, že potomek „zdědí“ špatné vlastnosti.

6 Implementace

Vytvořený program je poměrně rozsáhlý. V následující kapitole si popíšeme jen jeho nejdůležitější části.

6.1 Vybraná platforma

Microsoft .NET (zkráceně .NET) je zastřešující název pro technologie od společnosti Microsoft, které umožňují vytváření aplikací pro různé typy zařízení od mobilu, až po osobní počítače (viz. [35]). Hlavní komponentou je .NET Framework, prostředí pro běh aplikací. Z hlediska vývojářů nabízí .NET Framework komplexní a konzistentní programovací model, který umožňuje vytvářet kvalitní aplikace jak z vizuálního hlediska, tak i z hlediska bezpečnosti a bezproblémovosti. Mezi výhody platformy .NET patří:

- .NET nedefinuje žádný jazyk, ve kterém by se implicitně měly programy psát. Kód napsaný v jakémkoliv z dostupných jazyků (Visual Basic, Delphi, C#, J#, F#) se překládá do mezijazyka nazvaného Common Intermediate Language. Aplikace se pak při spuštění překládá z „mezijazyka“ na nativní kód daného zařízení, na kterém je program spuštěn.
- .NET je profesionální, komerčně podporovaná platforma. Tato skutečnost nám přináší jistotu, že vypuštěné změny budou vždy důkladně otestovány a v případě potíží se máme vždy na koho obrátit.
- Pro vývoj aplikací Microsoft vytvořil vývojové prostředí Visual Studio. Visual Studio zjednodušuje celý proces vývoje od návrhu a tvorby, až po testování a nasazení.

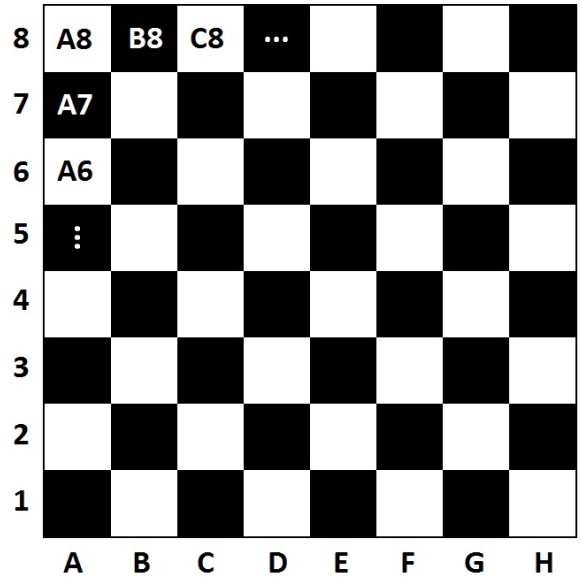
Mezi největší nevýhody bychom mohli zařadit skutečnost, že je .NET platforma primárně určena jen pro systémy Windows. Dnes už ale existují open source projekty implementující .NET i pro ostatní operační systémy (napr. MONO [36]).

Pro implementaci byl použit jazyk C# a .NET Framework verze 4.

6.2 Hrací deska

Jak už bylo řečeno, tato práce není zdaleka první, která se zabývá vytvořením počítačového programu simulujícího hru Dáma. Je více způsobů, kterými je možné jednotlivé části hry naprogramovat. Než se do psaní kódu pustíme, je nutné se rozhodnout, jakým způsobem budeme ukládat pozice jednotlivých figurek. Neexistuje žádná konvence, která by programátorům říkala, jakým způsobem by ve hře měla být například hrací deska reprezentována. Je plně na vůli autora, který ze způsobů si zvolí.

Způsobů je několik. Často se využívá reprezentace pomocí tzv. bitboardů [37]. Bitboard si můžeme představit jako číslo, jehož bitová délka je rovna počtu využívaných polí hrací desky (u hrací desky velikosti 8x8 se ve hře Šachy využívá všech 64 políček, v Dámě jen 32). Každá z pozic je reprezentována jedním bitem čísla. Nad tímto číslem pak provádíme bitové operace (OR, AND, XOR, atd.). Nevýhoda takového přístupu je, že jeden



Obrázek 18: Označení hrací desky

bit dokáže reprezentovat jen 2 stavy (například 0 = neobsazená pozice, 1 = obsazená). Tento nedostatek řeší například reprezentace hrací desky pomocí pole. Pole může být jak jednorozměrné, tak i dvourozměrné. Jednorozměrné pole je pro reprezentaci hráči desky postačující. Každá z pozic pole představuje jednu z pozic hrací desky. Výhodou dvourozměrného pole je to, že kopíruje strukturu desky a dotazování se na jednotlivé pozice může být pro někoho přirozenější.

Hrací deska je čtvercová síť, po které se hráči pohybují svými figurkami. Dvourozměrné pole si také můžeme představit stejným způsobem. Deska pro Anglickou dámu má 8x8 políček (Obrázek 18). Aby se ulehčila komunikace mezi jednotlivými hráči (nebo mezi hráči a rozhodčím), je každá z těchto políček pojmenovaná. Políčka na svislé ose „šachovnice“ jsou označeny čísly a políčka na vodorovné ose písmeny. Pokud se chceme odkázat na některou z pozic, musíme říci nejdříve, ve kterém sloupci se nacházíme a pak ve kterém řádku. Například políčko nacházející se v levém horním rohu Obrázku 18 bude pojmenované $A8$. Použitím dvourozměrného pole se budeme na jednotlivé pozice dotazovat stejným způsobem (sloupec v poli, řádek v poli). Tento přístup má i svoji nevýhodu. V případě Dámy budeme mít pozice v poli, které nebudou využívány.

Datový typ pole si můžeme také zvolit podle libosti (string, char, int, atd.). Pokud reprezentujeme hrací desku jako pole, je efektivní si typ hodnot, které se v poli budou ukládat zvolit tak, abychom byli schopni adresovat pomocí hodnot všechny druhy figurek, které se ve hře vyskytují. V Dámě jsou 4. Hráčův kámen, soupeřův kámen, hráčův král a soupeřův král. Místo primitivních datových typů si také můžeme vytvořit vlastní třídu, která bude reprezentovat jednotlivé figurky.

6.3 Třída „Piece“

Třída *Piece* reprezentuje jednotlivé figurky, které se na šachovnici nachází. Do 2D pole, které reprezentuje šachovnici, ukládáme instance této třídy (pole je tedy také typu Piece). Jednotlivé objekty, odvozené z této třídy uchovávají v sobě následující informace:

- vlastníka figurky,
- typ figurky,
- pozice figurky.

Vlastníkem figurky může být hráč 1 nebo hráč 2. Často se hráči rozlišují podle barev na černého a bílého hráče. Hráčem 1 je začínající hráč a hráčem 2 je jeho soupeř. Ne vždy se v Dámě používají jen černé a bílé figurky (mohou být také např. červené, nebo jakékoliv jiné barvy). V mém případě ještě v tuto chvíli nebylo jasné, jak bude grafické uživatelské rozhraní vypadat, proto jsou hráči pojmenováni jako $P1$ a $P2$. Toto odkazování na hráče je univerzálnější. Hráč P1 je vždy začínajícím hráčem nezávisle na tom, jaké barvy jsou jeho figurky.

Typ figurky definuje jeho vlastnost, jestli je nebo není králem. Hodnoty, které může figurka nabývat jsou: MAN a $KING$ (MAN je anglické označení pro kámen, který není králem).

Pozice figurky určuje, na kterém políčku desky se daná figurka nachází. Tento parametr je typu POINT, což je uspořádaná dvojice souřadnic X a Y. Souřadnice X představuje řádky v poli a souřadnice Y její sloupce.

Pokud na některé z pozic není žádná figurka, je tato pozice v poli naplněná hodnotou $null$.

6.4 Reprezentace tahu

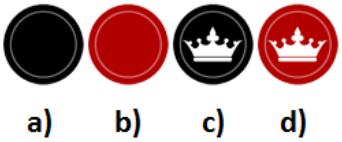
Tahy jsou taktéž reprezentovány vlastní třídou MOVE. Parametry této třídy jsou:

- seznam pozic,
- typ tahu,
- figurka, která tah provádí.

Seznam pozic obsahuje souřadnice polí, které se tah týká. Na nulté pozici nalezneme vždy výchozí umístění figurky. Na dalších pozicích pak nalezneme souřadnice polí, kterými figurka putuje k dosažení svého cíle. V jednom tahu může figurka putovat skrze více pozic v případě, že provádí vícenásobný skok.

Typ tahu určuje, jestli je tah skokem nebo nikoliv. Tento parametr je důležitý z hlediska dodržování pravidla, které nám říká, že pokud tah existuje, musí být vykonán.

Obrázek 19: Grafické rozhraní hrací desky



Obrázek 20: Grafické reprezentace figurek, a) kámen hráče 1, b) kámen hráče 2, c) král hráče 1, d) král hráče 2

6.5 Grafické uživatelské rozhraní

Jedním z požadavků této diplomové práce bylo vytvoření hry Dáma s grafickým uživatelským rozhraním. Dvourozměrné pole, které reprezentuje hrací desku je jen tzv. „backendovým“ zastoupením této struktury. Frontend, tedy grafické rozhraní této struktury, je znázorněno na Obrázku 19.

Hrací deska je poskládána z instancí třídy *PictureBox*. PictureBox je prvek, který umožňuje zobrazit obrázek. Na pozadí každého z těchto prvků je obrázek, který zobrazuje tmavé nebo světlé hrací pole v závislosti na jeho umístění. Na popředí se vykresluje grafické ztvárnění jednotlivých figurek (Obrázek 20).

6.6 BoardController

Důležitou částí programu je komponenta BoardController (zkráceně kontrolér). Ukládá důležité informace jako jsou: seznam hráčů, hráč na tahu, číslo tahu a také řídí celý průběh hry.

Po zahájení nové hry kontrolér vytvoří podmínky pro novou „partičku“. Nejdříve se inicializuje hrací deska, čímž se vytvoří dvourozměrné pole s výchozím rozestavěním. Pak se vyzve frontend hrací desky, aby vykreslil aktuální stav uložený v backendu (o kontroléru můžeme říci, že je jakousi spojovací vrstvou mezi front a backendem). Dalším krokem je vytvoření a uložení jednotlivých hráčů. Vzápětí kontrolér nastaví hráče 1 jako hráče, který je na tahu a vyzve ho k vykonání tahu. Hráč vrátí tah, který chce provést. Pokud hráč na tahu nemá žádný další tah, znamená to vítězství pro soupeře a hra končí. V opačném případě kontrolér zařídí aplikování změn, které tahem nastaly.

6.7 Hráči

Program umožňuje zvolit si z několika typů her:

- lidský hráč vs. lidský hráč,
- lidský hráč vs. UI,
- UI vs. UI.

V závislosti na tom, jaký typ hry si zvolíme, vytvoří kontrolér příslušné hráče.

6.7.1 Lidský hráč

Interakce lidského hráče probíhá skrze události, které hráč vyvolává na straně frontendu. Pokud je na tahu lidský hráč, kontrolér mu umožní pomocí události *PieceSelected* vybrat figurku (resp. PictureBox, na kterém je figurka vykreslena), kterou chce táhnout. Po kliknutí na figurku ve frontendu, kontrolér vypočítá, jaké tahy je možné provést danou figurkou a zobrazí je. Kontrolér také slouží pro zamezení vykonávání takových tahů hráčem, které nejsou v souladu s pravidly. Pokud se hráč snaží dopravit svoji figurku na pozici, která nebyla po označení figurky zvýrazněna, nebude tah akceptován. Rovněž pokud hráč chce vykonat jiný tah než skok ve chvíli, kdy skok vykonat může, nebude taková akce povolena.

Vytvoření tahu lidským uživatelem je velmi jednoduché. Stačí kliknout na figurku, se kterou chceme tah vykonat a následně kliknout na pozici (resp. pozice, pokud jde o skok) kam chceme figurku dopravit. Tah bude předán kontroléru, který ho následně zpracuje.

6.8 UI

V případě, že zvolíme hru, ve které vystupuje agent UI, je nutné zadat dodatečné parametry jako: úroveň agenta a algoritmus, který bude využívat na generování tahů. Úroveň, na jaké hraje lidský hráč závisí na jeho zkušenostech a dovednostech. V případě UI si můžeme zvolit ze 3 obtížností: *začátečník*, *pokročilý* a *profesionál*. Jednotlivé úrovně se od sebe liší v počtu tahů (v hloubce do jaké výpočty probíhají), které je algoritmus schopny dopředu vypočítat (Tabulka 4).

Z hlediska generování tahu si můžeme vybrat z algoritmů:

- naivní algoritmus,

Úroveň obtížnosti	Hloubka výpočtu
Začátečník	2
Pokročilý	4
Profesionál	6

Tabulka 4: Hloubka výpočtu jednotlivých úrovní obtížností

- Minimax,
- Negamax,
- GA.

Hlavním cílem agenta UI je vrátit legální tah. Tento tah by měl být, pokud možno, nejlepší. Každý z agentů UI se skládá ze dvou hlavních částí. Z generátoru tahů a ohodnocovací funkce. Všechny z vyjmenovaných algoritmů jsou schopné využívat stejné ohodnocovací funkce. Kvůli rozmanitosti jsem pro každý z algoritmů vytvořil vlastní způsob ohodnocování.

Funkcí generátoru tahů je vytvořit seznam všech možných tahů a ohodnocovací funkce pak tyto tahy ohodnotí. Určí, který z nich je nejlepší. Vybraný tah se pak vrátí kontroléru ke zpracování.

6.8.1 Naivní algoritmus

Generátor tahů pro naivní algoritmus je velice jednoduchý. Pro každou z figurek patřící agentovi UI se vytvoří tahy, které s nimi lze provést. Algoritmus je blíže popsaný v kapitole 3.2.

„Kvalitu“ tahů určí ohodnocovací funkce. Ohodnocovací funkce vyhodnotí stav z pohledu UI po vykonání jednotlivých tahů. Vybrán bude ten z tahů, který po provedení dostane hráče do pro něho nejvýhodnější situace. Ohodnocovací funkce vypočítá sílu hráče na tahu i soupeřovu sílu. Tento výpočet probíhá na základě materiálního ohodnocení, tj. každá figurka na hrací desce je ohodnocena podle tabulky 5. Pseudokód ohodnocovací funkce, kterou naivní algoritmus využívá je k vidění ve Výpisu 4.

Druh figurky	Hodnota
kámen	5
král	7

Tabulka 5: Hodnoty figurek

```
int Ohodnot(Stav s, Hrac hracNaTahu)
{
    for (i = 0; i < vsechnyFigurky.pocet; i++)          /* Iterace skrze všechny figurky */
    {
        if (vsechnyFigurky[i].majitel == hracNaTahu)
        {
            if (vsechnyFigurky[i].typ != KRAL)
            {
            /* Pokud patří figurka hráči na tahu a není králem, hráč získá 5 bodů */
                skoreHraceNaTahu += 5;
            }
            else
            {
            /* Pokud patří figurka hráči na tahu a je králem, získá 7 bodů */
                skoreHraceNaTahu += 7;
            }
        }
        else
        {
            if (vsechnyFigurky[i].typ != KRAL)
            {
            /* Pokud patří figurka soupeři a není králem, soupeř získá 5 bodů */
                skoreSoupere += 7;
            }
            else
            {
            /* Pokud patří figurka soupeři a je králem, soupeř získá 7 bodů */
                skoreSoupere -= 7;
            }
        }
    }
    return skoreHraceNaTahu - skoreSoupere;
}
```

Výpis 4: Pseudokód ohodnocovací funkce naivního algoritmu

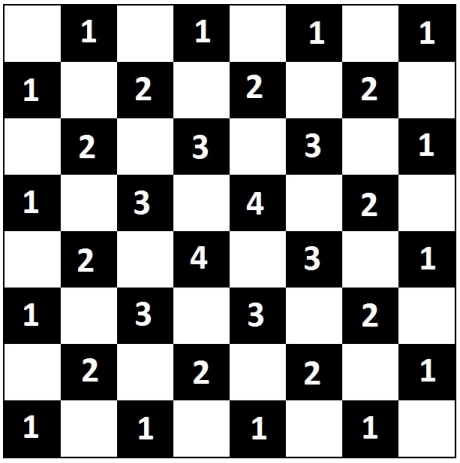
Výsledné skóre dostaneme odečtením soupeřového skóre od skóre hráče na tahu.

6.8.2 Minimax

Generování tahu v případě algoritmu Minimax probíhá několik tahů dopředu. Navíc se algoritmus na hru už „nedívá“ jen z vlastního pohledu, ale i z pohledu soupeře. To, kolik tahů dopředu jsme schopni vygenerovat a také kvalita ohodnocovací funkce nám udávají, jak dobře bude agent využívající tento algoritmus hrát.

Druh figurky	Hodnota
kámen	5
kámen, který je krok od toho aby se stal králem	7
král	10

Tabulka 6: Hodnoty figurek (rozšířené)



Obrázek 21: Ohodnocení králů podle jejich pozice

Ohodnocovací funkce Minimaxu je poměrně jednoduchá a podobná té z naivního algoritmu. I zde k ohodnocení tahu využíváme materiálního ohodnocení, ale v tomto případě jsme ho mírně rozšířili. Figurky se hodnotí podle tabulky 6.

Takto navržené materiální ohodnocení zvýhodňuje figurky, které jsou blízko tomu, aby se staly králem. Počtem 7 bodů bude ohodnocena každá figurka, která je z pohledu hráče 1 v šesté řadě a z pohledu hráče 2 v druhé řadě. Minimax tedy využívá podobnou ohodnocovací funkci jako naivní algoritmus (popsáno ve Výpisu 4). Rozdíl je v přidělování bodů hráčům. Jakým způsobem se body přidělují znázorňuje Výpis refPseudokodMinimaxEvaluate.

Kromě materiálního ohodnocení využívá zde Minimax i tzv. pozičního ohodnocení. Pokud je figurka králem, je hodnocena i na základě pozice, na které se nachází. Poziční hodnocení jsme si popsali v kapitole 3.6. Jak jsou jednotlivá pozice ohodnoceny je vidět na Obrázku 21.

```
int calculateValue(Figurka figurka)
{
    /* Ohodnoceni z pohledu hráče P1 */
    if (figurka.majitel == P1)
    {
        if (figurka.typ != KRAL)
        {
            /* Pokud figurka není králem, ale je blízko tomu, aby se jím stal, je hodnocena 7 body
               */
            if (figurka.pozice == sestaRada)
            {
                hodnotaFigurky = 7;
            }
            /* Pokud figurka není králem a není blízko tomu, aby se jím stal, je hodnocena 5 body */
            else
            {
                hodnotaFigurky = 5;
            }
        }
        else
        {
            /* Pokud je figurka králem, je hodnocena 10 body a vynásobena hodnotou pozice, na
               které stojí */
            hodnotaFigurky = 10 * bodyZaPozici(figurka.pozice);
        }
    }
    /* Ohodnoceni z pohledu hráče P2 */
    else
    {
        if (figurka.typ != KRAL)
        {
        /* Pokud figurka není králem, ale je blízko tomu aby se jím stal, je hodnocena 7 body */
            if (figurka.pozice == druhaRada)
            {
                hodnotaFigurky = 7;
            }
            /* Pokud figurka není králem a není blízko tomu, aby se jím stal, je hodnocena 5 body */
            else
            {
                hodnotaFigurky = 5;
            }
        }
        else
        {
            /* Pokud je figurka králem, je hodnocena 10 body a vynásobena hodnotou pozice, na
               které stojí */
            hodnotaFigurky = 10 * bodyZaPozici(figurka.pozice);
        }
    }
    return hodnotaFigurky;
}
```

Výpis 5: Pseudokód ohodnocení jednotlivých figurek v algoritmu Minimax

6.8.3 Negamax

V kapitole 3.4 jsme si už Negamax představili a řekli jsme si o něm, že se jedná jen o alternativní zápis algoritmu Minimax. Důvod, proč jsou využívané oba algoritmy, je pouze demonstrační. Největší rozdíl mezi nimi je v ohodnocovací funkci, která je v případě Negamaxu sofistikovanější. Využívá k ohodnocení stavu sadu různých parametrů. Výsledné skóre se počítá jako součet všech hodnot, které jednotlivé parametry vrací.

Může se stát, že několik tahů bude ohodnoceno stejným skóre. Aby algoritmy nebyly jasně deterministické, je zde zaveden i faktor náhody. V případě, že se dva tahy ukážou jako stejně výhodné (byly ohodnoceny stejným skóre), náhodně se vybere jeden z nich.

6.8.3.1 Materiální převaha: V podstatě se jedná o materiální ohodnocení stavu podobný tomu, který je využívaný naivním algoritmem i Minimaxem. Hráč je hodnocen třemi body za každou figurku, která není králem a sedmi body za každého krále. Z jeho skóre se pak odečte stejný počet bodů za každou soupeřovu figurku (-3 za každý kámen, -7 za každého krále).

6.8.3.2 Ovládání středu pole I: Hráč na tahu získává 4 body za každou z pozic $B6, D6, F6, C5, E5, G5, B4, D4, F4, C3, E3$ a $G3$, která je obsazena jeho figurkou.

Strategie, kterou začátečníci často využívají, je ochránění svých figurek tím, že je dostanou na krajní pozice. Dáma ale není hra, kterou se vyplatí hrát defenzivně. Jedno z pravidel hry říká, pokud hráč má dostupný skok, musí ho provést. Pokud se hráč snaží hrát defenzivně, znamená to, že nechá soupeře připravit si vlastní taktiku. Ten defenzivního hráče obětováním své vlastní figurky může dostat do nevýhodné situace. Kvůli vynuceným skokům je skoro nemožné vytvořit obranu, která by se časem nezhroutila. Proto je pro ofenzivní způsob hraní klíčové ovládání středu pole.

6.8.3.3 Ovládáni středu pole II: Hráč na tahu získává 3 body za všechny pozice, ze kterých se dalším tahem může dostat na jednu z pozic $B6, D6, F6, C5, E5, G5, B4, D4, F4, C3, E3$ nebo $G3$. Parametr zvýhodní figurky, které by mohly ovládnout střed hracího pole.

6.8.3.4 Ovládání středu pole králem: Hráč na tahu obdrží 10 bodů za každou z pozic $B6, D6, F6, C5, E5, G5, B4, D4, F4, C3, E3$ a $G3$ na které má vlastního krále.

6.8.3.5 Parametr postupu: Hráč na tahu obdrží 5 bodů za každou figurku hráče, která není králem a je v šesté nebo sedmé řadě z pohledu hráče 1, nebo v druhé a třetí řade z pohledu hráče 2. Tento parametr nutí hráče postupovat vpřed. Král je silná figurka, která často rozhoduje o výsledku hry. Obecně řečeno, hráč, který má více králů, má větší šanci vyhrát. Proto je důležité zvýhodnit figurky, které jsou blíže tomu, aby se staly králem.

6.8.3.6 Back row bridge: Hráč na tahu získává 10 bodů, pokud jeho zadní řada je chráněná, tzv. back row bridgem. Figurky, které se nachází na pozicích $C1$ a $G1$ z pohledu hráče 1 a na pozicích $B8$ a $F8$ z pohledu hráče 2, tvoří tzv. back row bridge. Konstelaci,

která je z defenzivního hlediska velmi důležitá. Figurky na těchto pozicích jsou schopné zabránit proniknutí soupeře do svých zadních řad.

6.8.3.7 Volný pohyb: V případě, že se na hrací desce nachází méně než 24 figurek a počet figurek je lichý, obdrží hráči 5 bodů za každou svoji volnou figurku.

Tento parametr zvýhodňuje takovou situaci, která umožňuje volnější pohyb na hrací desce. Čím méně figurek nalezneme na desce, tím je snazší se zbylými figurkami pohybovat. Lichý počet figurek na desce znamená, že jeden z hráčů má početní převahu a alespoň 1 z jeho figurek je nekrytá a může se volně pohybovat.

6.8.3.8 Mobilita: Hráč obdrží 2 body za každou pozici, na kterou může dopravit jednu ze svých figurek bez toho, aniž by tato figurka byla ohrožena soupeřem.

6.8.3.9 Ohrožení: Hráč obdrží 3 body za každou pozici, na kterou může aktivní hráč dopravit jednu ze svých figurek, přičemž tím ohrozí jednu ze soupeřových figurek.

6.8.3.10 Parametr vítězství: Hráč obdrží 10 000 bodů, pokud jeho soupeř nemá na hrací desce žádné figurky.

6.8.4 GA

Pokud si hráč zvolí za soupeře GA, bude hrát ve skutečnosti s UI, která jako generátor tahů využívá algoritmu Negamax. Ohodnocovací funkce je zde také stejná jako v případě Negamaxu. Rozdíl je v tom, že jednotlivým parametrům, podle kterých Negamax ohodnocuje tahy, jsou přiřazené váhy. Tyto váhy určují, jaký důraz je kladen na parametr, ke kterému se váže. Váhy jsou reprezentovány celými čísly z intervalu -1000 až $+1000$. Čím větší kladné číslo je parametrům přiděleno, tím větší je jeho váha. Váhy násobí skóre, které je podle jednotlivých parametrů vypočítané. Výsledné skóre se pak počítá podle vzorce:

$$skore = (vaha_1 * parametr_1) + (vaha_2 * parametr_2) + \ldots + (vaha_n * parametr_n).$$

Určit důležitost jednotlivých parametrů pomocí vah může být náročným úkolem. Správně odhadnout rovnováhu mezi parametry vyžaduje zkušenosti a dobré analytické myšlení. Ani tehdy ale není zaručeno, že dosáhneme požadovaného výsledku. Najít nejlepší řešení bychom mohli tehdy, pokud bychom vyzkoušeli každou z možností a podrobili bychom je testu, který by určil kvalitu řešení. Takový postup by byl časově i výpočetně velmi náročný. Dalším řešením, kterým lze váhy vybalancovat, je použít pro tento úkol GA.

GA zde slouží „pouze“ jako nástroj pro nalezení správné rovnováhy mezi váhou jednotlivých parametrů. Pomocí GA jsme schopni vygenerovat váhy, vyzkoušet jejich účinek a vyhodnotit jejich dopad. Princip, kterým GA vytváří váhy je následující. Jako vstupní parametr algoritmu musíme zadat počet generací, které má GA vytvořit. Tato

hodnota slouží k tomu, aby se algoritmus po jisté době zastavil. Na začátku algoritmu se vytvoří populace. Populaci tvoří agenti UI (implicitně se vytváří 30 jedinců). Jak již bylo zmíněno, tihle hráči využívají algoritmu Negamax k určení následujícího tahu a parametru k ohodnocení tahů. Na začátku se pro každý z těchto parametrů náhodně přiřadí váhy. Každý jedinec z populace bude mít vygenerovanou vlastní sadu vah. Tímto se ukončí inicializace populace (nulté generace).

V tomto bodě má algoritmus vytvořený předem specifikovaný počet hráčů, každý z vlastní sadou vah. V dalším kroku se algoritmus snaží zjistit, který z těchto hráčů je schopen hrát nejlépe, resp. která z vygenerované sady vah má nejlepší dopad na „výkon" hráče. K tomu se využívá tzv. round robin turnaje, který jsme si popsali v části 5.2. Turnaje se účastní všichni hráči a turnaj funguje podle systému „každý s každým". V případě, že máme populaci sestavenou z n jedinců, můžeme si počet her, které se budu hrát, vypočítat jako variace k-té třídy z n prvků, tedy podle vzorce.

$$V(n,k) = \frac{n!}{(n-k!)}$$

Pro populaci, kterou tvoří 30 jedinců, bude výpočet vypadat následovně: $V(2,30) = \frac{30!}{(30-2!)} = \frac{30!}{28!} = 870$. Celkový počet her, které odehraje 30 členná skupina je tedy 870. Odehrání jedné hry trvá lidským hráčům řádově několik desítek minut. Počítač samozřejmě dokáže odehrát hru mnohanásobně rychleji. Může se ale stát, že hráči se „zaseknou" ve stavu, kdy jeden „honí" druhého po hrací desce. Taková hra by trvala nekonečně dlouho a z tohoto důvodu je nutné nadefinovat nějakou ukončovací podmínku. V našem případě je ukončovací podmínkou maximální počet tahů. Pokud hráči nejsou schopni dohrát hru v předem definovaném počtu tahu (implicitně 100 tahů), bude hra ukončena a vyhodnocena jako remíza. Pokud jeden z hráčů je schopný vyhrát před dosažením maximálního počtu tahů, přiřadí mu účelová funkce jeden bod.

Po ukončení turnaje bude na základě dosažených bodů vybrána polovina populace, která založí další generaci. Druhá polovina populace bude smazána a nahrazena novými jedinci. Noví jedinci vznikají křížením zbylých hráčů a pomocí mutace. Ze zbylých hráčů se náhodně vyberou dva rodiče. Z rodičů se opět náhodně vyberou váhy, které budou zkopírovány do nově vytvořeného jedince. Tímto způsobem vznikne potomek, který v sobě uchovává informace svých rodičů. Aby se zabránilo tomu, že se jednotlivé váhy v každé generaci budou opakovat, v nově vzniklém jedinci probíhá i mutace. Mutace náhodně vybere některou z vah a změní ji o $\pm 25\%$. Počet mutací, které nastanou u každého jedince, je opět možné nastavit.

Nově vzniklou generaci budou tvořit jedinci z předchozí generace a stejný počet jedinců, kteří byli od nich „odvozeni". Váhy nejlepšího z nich jsou použity algoritmem Negamax. Ukázku vygenerovaných vah 20. generace můžeme vidět v Tabulce 7.

- Piece = materiální převaha
- CentI = ovládání středu pole II
- CentII = ovládání středu pole II

Fitness	Piece	CentI	CentII	Kcent	ADV	BRB	MOVE	MOB	THRET	VTY
42	680	859	-271	-10	ADV	553	-642	-515	-867	-702
40	510	859	-271	-10	ADV	553	-802	-515	867	-702
39	766	189	-433	21	553	-872	-515	-746	109	-443
38	680	341	110	534	553	-974	-515	-867	317	-570
37	887	385	-34	200	553	-1016	-515	-169	202	-570
...	...	...	...	...	...	...	...	...	...	...

Tabulka 7: Ukázka vytvořených vah (20. generace)

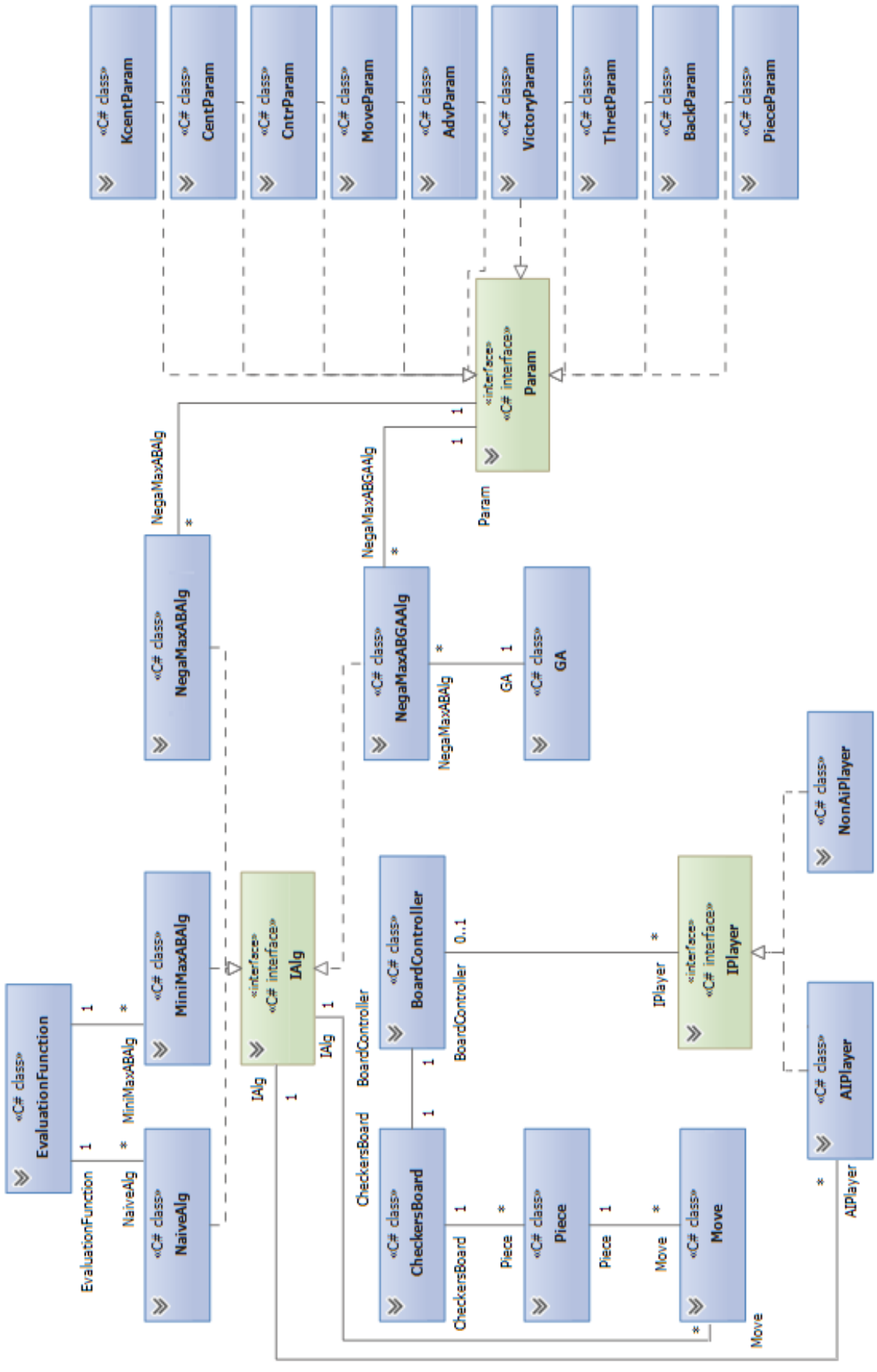
- Kcent = ovládání středu pole králem
- Adv = parametr postupu
- BRB = back row bridge
- MOVE = volný pohyb
- MOB = mobilita
- THRET = ohrožení
- VTY = parametr vítězství

6.8.5 Nápověda

Lidským hráčům je umožněno využívání nápovědy. Nápověda radí hráči, který z dostupných tahů je pro něj nejvýhodnější. Systém nápovědy využívá ke své predikci algoritmu Minimax. Po kliknutí na tlačítko nápověda proběhne výpočet, který vrátí nejlepší tah a ten se vzápětí zobrazí v podobě textového řetězce.

6.8.6 Třídní diagram

Třídní diagram, který znázorňuje propojení mezi jednotlivými třídami a jejich závislosti, je vidět na Obrázku 22.



Obrázek 22: Třídní diagram

7 Dosažené výsledky

7.1 Porovnání úrovní obtížností

V této části se zaměříme na to, jak si jednotlivé algoritmy vedou na různých úrovních obtížnosti. Pro připomenutí, úrovně se liší hloubkou, do které výpočet probíhá. Hloubku úrovně zobrazuje Tabulka 4 z kapitoly 6.8. Test je formou turnaje. Všichni hráči využívají stejný algoritmus pro generování tahů a stejnou ohodnocovací funkci. Liší se pouze v hloubce výpočtu. Hráči proti sobě hrají vždy 30 partiček. Hra se vyhodnotí jako remíza, pokud žádný z hráčů nevyhraje během 200 tahů.

7.1.1 Negamax začátečník vs. Negamax začátečník

Výsledky turnaje zachycuje Obrázek 26. Lze z něj vyčíst, že oba dva hráči jsou schopni vyhrávat. Hráči jsou na stejné úrovni a očekává se vyrovnaná hra. Graf ale ukazuje, že jeden z hráčů má převahu, což může být způsobeno nízkým počtem vzorků.

7.1.2 Negamax začátečník vs. Negamax pokročilý

Zde očekáváme, že hráč hrající na pokročilé úrovni bude jednoznačně dominovat. Tento předpoklad nám potvrzuje Obrázek 27. Hráč s větší hloubkou výpočtu ani jednou neprohrál. Poměrně velký počet remízy má na svědomí skutečnost, že je hra přerušena po 200 tazích a vyhodnocena remízou nezávisle na tom, který z hráčů má převahu.

7.1.3 Negamax začátečník vs. Negamax profesionál

Hráč na pokročilé úrovni už hrál bez jediné porážky. Od úrovně profesionál očekáváme, že jeho dominance bude více zřetelná. Výsledky turnaje jsou k vidění na Obrázku 28. Hráč s úrovní profesionál také neprohrál ani jednu ze svých her. Počet remíz se také mírně snížil, takže je zde vidět jisté zlepšení. Velký rozdíl byl zaznamenán v počtu tahů, které hráči potřebovali k dosažení výhry (Obrázek 29). Průměrný počet tahů, které hráč k výhře potřebuje, je u hráče na pokročilé úrovni 48, u hráče na profesionální úrovni 43. Průměr je zde počítán jenom z partiček, které skončily výhrou, nikoliv remízou.

7.2 Porovnaní algoritmů

Ve hře máme 3 hlavní typy algoritmů. Naivní algoritmus, Minimax a Negamax. Minimax a Negamax jsou ekvivalentní algoritmy, které se liší jen zápisem, využívají ale různých ohodnocovacích funkcí. Proto je jejich „herní vyspělost“ odlišná. Zde podrobíme všechny tři algoritmy testu abychom zjistili, který z nich umí nejlépe hrát Dámu. U naivního algoritmu nelze vybrat úroveň obtížnosti. Počítá se jen jeden tah dopředu. Minimax a Negamax budou nastaveny na začátečnickou úroveň obtížnosti (výpočet 2 tahy dopředu).

7.2.1 Naivní algoritmus vs. Minimax

Výsledek třiceti her zachycuje Obrázek 24. Test dopadl podle očekávání. Naivní algoritmus nedokáže konkurovat Minimaxu a ve třiceti kolech nevyhrál ani jednou. Všechny hry vyhrál Minimax a nenastala zde žádná remíza.

7.2.2 Naivní algoritmus vs. Negamax

Naivní algoritmus ani v tomto případě nebyl schopný jediné výhry nebo remízy. Výsledky testu zachycuje Obrázek 25. Naivní algoritmus se podle očekávání jeví jako nejslabší v porovnání s Minimaxem a Negamaxem.

7.2.3 Minimax začátečník vs. Negamax začátečník

Tyto algoritmy jsou si ekvivalentní a oba z nich budou hrát na úrovni začátečníka. O jejich schopnostech tedy rozhoduje pouze ohodnocovací funkce, kterou využívají. Výsledek turnaje je vidět na Obrázku 23. Navzdory tomu, že algoritmus Negamax využívá komplexnější způsob ohodnocování, dokázal nad Minimaxem vyhrát pouze třikrát a další čtyři hry skončily remízou. Zbytek ze třiceti her vyhrál Minimax. Pravděpodobným důvodem, proč Negamax hraje hůř, je některý z parametrů, který je špatně navržený, nebo je na něj kladen příliš velký důraz. Je důležité, aby tyto parametry byly v rovnováze. Pokud některý z parametrů bude zvýhodněn vysokým počtem bodů, bude to pro algoritmus znamenat, že bude ovlivněn hlavně tímto parametrem. To může mít negativní dopad na jeho schopnost hrát.

7.3 GA

GA jsme zde využili na vygenerování vah. Pokud by byla každá z vah rovna 1, znamenalo by to, že by byl tento algoritmus identický s Negamaxem. I zde je totiž využito ke generování tahu právě Negamax. Rozdíl je opět v ohonocovací funkci (popsaná v sekci 6.8.4). UI využívá GA „jen“ k nalezení rovnováhy mezi jednotlivými parametry. Vstupní parametry pro GA jsme si zvolili následovně:

- velikost populace = 30 hráčů,
- mutační faktor = 10,
- počet generací 30.

Počet generací jsme vytvářeli postupně. Každá pátá generace byla podrobena testu. Testování je časově náročný proces, proto se netestovala každá generace. Výsledek testu můžeme vidět na Obrázku 30.

Z údajů můžeme vyčíst, že nultá generace nebyla schopna vyhrát ani jednu hru. Nultou generací jsou váhy, které byly náhodně vygenerovány a nebyly ještě podrobeny žádnému výběru. Výsledek je podle očekávání. Náhodným zvolením vah bez jakékoliv optimalizace, algoritmus nevyhrává. Pátá generace je generací, která už prošla několika

výběry a již dosahuje lepších výsledků. Hráč pomocí GA dokázal už po páté generaci zlepšit svoji hru a byl schopen hrát vyrovnaně s Negamaxem. Znatelné zlepšení přinesla až 20. generace, kde hráč využívající váhy jasně dominuje. Následující generace přinesly mírné zhoršení. To, že se počet výher výrazně nezvyšuje po každé generaci, ale střídavě se mírně zlepšuje a zhoršuje, může mít za následek skutečnost, že jsme použili vysoký mutační faktor. Pokud je toto číslo vysoké, algoritmus rychleji prohledává prostor možných řešení. Může se ale stát, že po nalezení optimálního řešení se od něj opět vzdálí.

Mutační faktor určuje, jak velkou část z prostoru možných řešení budeme prohledávat. Pokud prohledáváme větší část, znamená to, že se vzdalujeme od doposud nalezeného nejlepšího řešení. Předchází se tím tomu, aby se algoritmus „zasekl“ u lokálního optima. Zvolením nižšího mutačního faktoru by se prodloužila doba potřebná k tomu, abychom se dopracovali k optimálnímu řešení. Mohlo by se také stát, že po nalezení lokálního optima by algoritmus v jeho blízkém okolí lepší řešení už nenasel a nikdy by se ke globálnímu optimu nedopracoval. Naopak zvolením vysokého mutačního faktoru můžeme optimální řešení nalézt rychleji, ale také ho opětovně ztratit, protože se v tomto případě snažíme prohledat širší okolí.

Na Obrázku 31 můžeme vidět, jaký byl počet výher nejlepšího hráče v jednotlivých generacích. Tato hodnota postupně klesá, což může mít za důsledek zvyšující se konkurenceschopnost jedinců v rámci stejné generace. Kolem patnácté generace se hodnota ustálí a pohybuje se kolem průměru 35 výher z celkového počtu 58 her, které musí každý jedinec odehrát.

8 Závěr

Jedním z cílů této diplomové práce bylo vytvořit aplikaci s umělou inteligencí počítače pro hraní deskové hry Dáma. Tento cíl byl dosažen a algoritmů pro umělou inteligenci se nám povedlo vytvořit hned několik. Nalezneme zde funkční implementace algoritmů Minimax, Negamax i naivního algoritmu. Bylo vytvořeno i několik druhů funkcí na ohodnocení pozic. Vytvořené algoritmy jsme podrobili testům. Ty prokázaly, že každý z nich je schopný hrát Dámu. Úroveň jejich schopností se liší. Naivní algoritmus se podle očekávání ukázal jako nejslabší z trojice algoritmů. Překvapivě dominantním byl algoritmus Minimax, který svoji jednoduchou ohodnocovací funkcí předčil Negamax.

Navzdory tomu, že jsme k vytvoření ohodnocovací funkce pro Negamax využili stejných parametrů, jakých využíval například neporazitelný Chinook, nebyl schopný nad algoritmem Minimax vyhrávat. Ten přitom jako ohodnocovací funkci používal pouze kombinaci materiálního a pozičního ohodnocení. Důvodem může být jak chyba v implementaci některého z parametrů, tak i skutečnost, že jednotlivé parametry nejsou ve správné rovnováze. Zde lze vidět, že je velice obtížné pro člověka pouhým odhadem tuto rovnováhu nalézt. Právě proto se často využívají metaheuristické algoritmy, kterým je i GA.

GA jsme zde využili k tomu, abychom hru Negamaxu vylepšili. Po několika generacích dokázal Negamax vyhrávat nad Minimaxem. Můžeme tedy prohlásit, že pomocí GA jsme schopni hru Negamaxu zlepšit. Kvůli nízkému počtu vytvořených generací ale nemůžeme jednoznačně říci, jestli je algoritmus schopný vytvořit hráče, kterého by Minimax nedokázal porazit. Přesný počet generací, které jsou zapotřebí k tomu, abychom se dopracovali k takovému výsledku, nelze odhadnout. Z odborné literatury se ale můžeme dočíst, že jsou to často stovky až tisíce generací. Kvůli nedostatečné výpočetní síle počítače použitého k testování jsme nebyli schopni takové množství generací vytvořit.

Roland Krutek

9 Reference

[1] Oficiální stránky České federace dámy: *http://www.damweb.cz*

[2] Mandziuk J., Kusiak M. Waledzik K.: *Evolutionary-based heuristic generators for checkers and give-away checkers*, Warsaw, Blackwell Publishing, 2007.

[3] Kusiak M. Waledzik K., Mandziuk J.: *Evolutionary Approach to the Game of Checkers*, Berlin, Springer Berlin Heidelberg, 2007. ISBN 978-3-540-71589-4

[4] Zapletal, M.: *Velká kniha deskových her*, Praha, Mladá Fronta, 1991. ISBN 80-204-0188-4

[5] Parlett, D.: *Oxford History of Board Games*, Oxford, Oxford University Press, 1999. ISBN 978-0192129987

[6] Freska znázorňující hru Senet: *http://digilander.libero.it/frangioila/egitto/nefertari/nefgiocaalsenet.jpg*

[7] Amfora znázorňující hru Ludus latrunculi: *http://erroso.blogspot.cz/2010/12/blog-post_27.html*

[8] Murray, H. J. R.: *A History of Board-games Other Than Chess*, Oxford, Oxford University Press, 1952. ISBN 978-0198274018

[9] McCorduck, P.: *Machines Who Think (2nd ed.)*, Natick, A. K. Peters, Ltd., 2004. ISBN 1568812051

[10] Turing, A.: *Computing Machinery and Intelligence*, Oxford, Oxford University Press, 1950.

[11] Sedláček, V.: *Umělá inteligence 1 (1. vyd.)*, Praha, Státní pedagogické nakladatelství, 1983.

[12] Bratko I.: *Prolog programming for artificial intelligence (3rd ed.)*, Harlow, Addison-Wesley, 2001. ISBN

[13] UI ve hrách: *http://psych.utoronto.ca/users/reingold/courses/ai/games.html*

[14] Stromy: *http://teorie-grafu.cz/zakladni-pojmy/stromy.php*

[15] Russell S.J., Norvig P.: *Artificial Intelligence. A modern approach.*, New Jersey, Prentice Hall, 1995. ISBN 0-13-103805-2

[16] Newell A., Shaw J. C., Simon H. A.: *Chess-playing programs and the problem of complexity*, New York, Springer New York, 1958. ISBN 978-1-4613-8716-9

[17] Flegr, J.: *Úvod do evoluční biologie. (2. vyd.)*, Praha, Academia., 2007. ISBN 978-80-200-1539-6

[18] Fogel, L.J., Owens, A.J., Walsh, M.J.: *Artificial Intelligence through Simulated Evolution*, Michigan, John Wiley & Sons, 1966. ISBN 0471265160

[19] Rechenberg, I: *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Stuttgart, Fromman-Holzboog, 1971. ISBN 9783772803734

[20] Schwefel , H. P.: *Numerical Optimization of Computer Models*, Basel, John Wiley and Sons Ltd., 1977. ISBN 0-471-09988-0

[21] Holland, J. H.: *Outline for a Logical Theory of Adaptive Systems*, New York, ACM New York, 1962.

[22] Haupt R.L., Haupt S. E.: *Practical Genetic Algorithms*, New York, Wiley-Interscience, 1998. ISBN 0471455652

[23] Chellapilla, K., Fogel D.: *Evolving an expert checkers playing program without using human expertise*, La Jolla, IEEE Transactions, 2001.

[24] Genetic Algorithms and Evolutionary Computation: *http://www.talkorigins.org/faqs/genalg/genalg.html*

[25] Programovatelné hradlové pole: *http://cs.wikipedia.org/wiki/Programovatelné_hradlové_pole*

[26] Assion, A. Baumert,T. Bergt, M.,Brixner, T., Kiefer, B., Seyfried, V., Strehle M., Gerber G.: *Control of chemical reactions by feedback-optimized phase-shaped femtosecond laser pulses*, Wessling, Deutsches Zentrum für Luft- und Raumfahrt, 1998.

[27] Kewley, R., Embrechts M.: *Computational military tactical planning system*, Piscataway, IEEE Press, 2002.

[28] Rizki, M., Zmuda, M. Tamburino, L.: *Evolving pattern recognition systems*, Piscataway, IEEE Press, 2002.

[29] Mitchell, M.: *An Introduction to Genetic Algorithms*, Massachusetts, MIT Press, 1996. ISBN 0-262-63185-7

[30] Struktura bílkovin: *http://cs.wikipedia.org/wiki/Bílkovina*

[31] Dorigo M., Stützle T.: *Ant colony optimization*, Massachusetts, MIT Press, 2004. ISBN 0262042193

[32] Kennedy, J., Eberhart, R.: *Neural Networks: Particle swarm optimization*, Washington, Bur. of Labor Stat., 1995. ISBN 0-7803-2768-3

[33] Godfrey C. Onwubolu, B. Babu V.: *New Optimization Techniques in Engineering*, New York, Springer, 2004. ISBN 978-3-540-20167-0

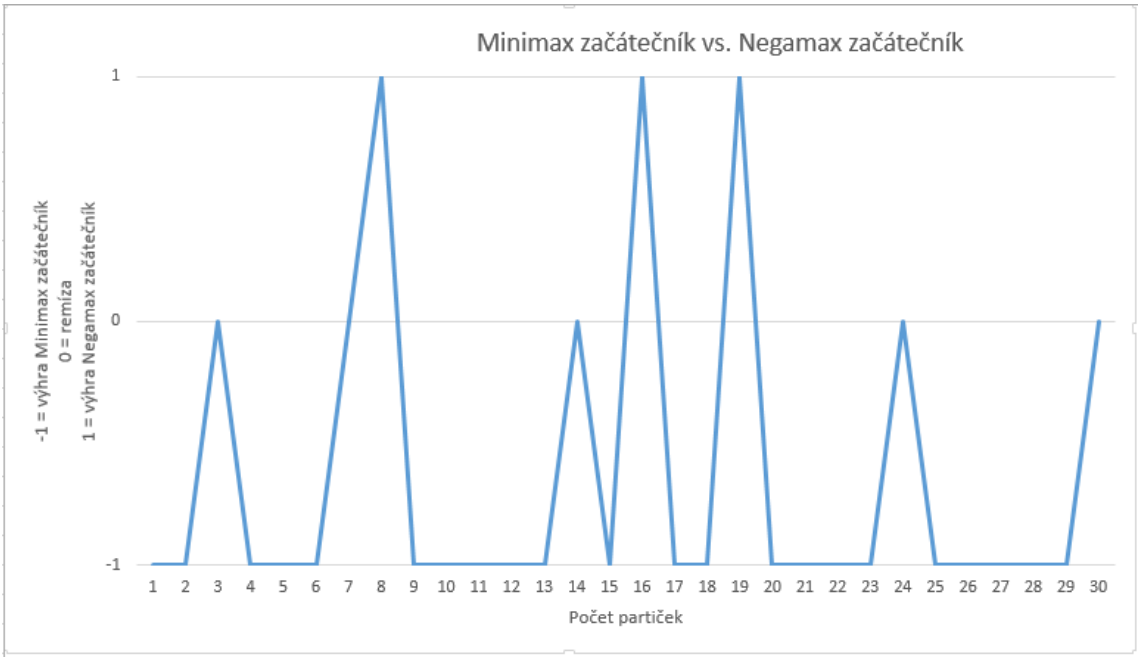
[34] Oplatková Z., Ošmera P., Šeda M., Včelař F., Zelinka I.: *Evoluční výpočetní techniky - principy a aplikace*, Praha, BEN-Technická literatura, 2008. ISBN 80-7300-218-3

[35] Microsoft .NET: *http://www.microsoft.com/net*

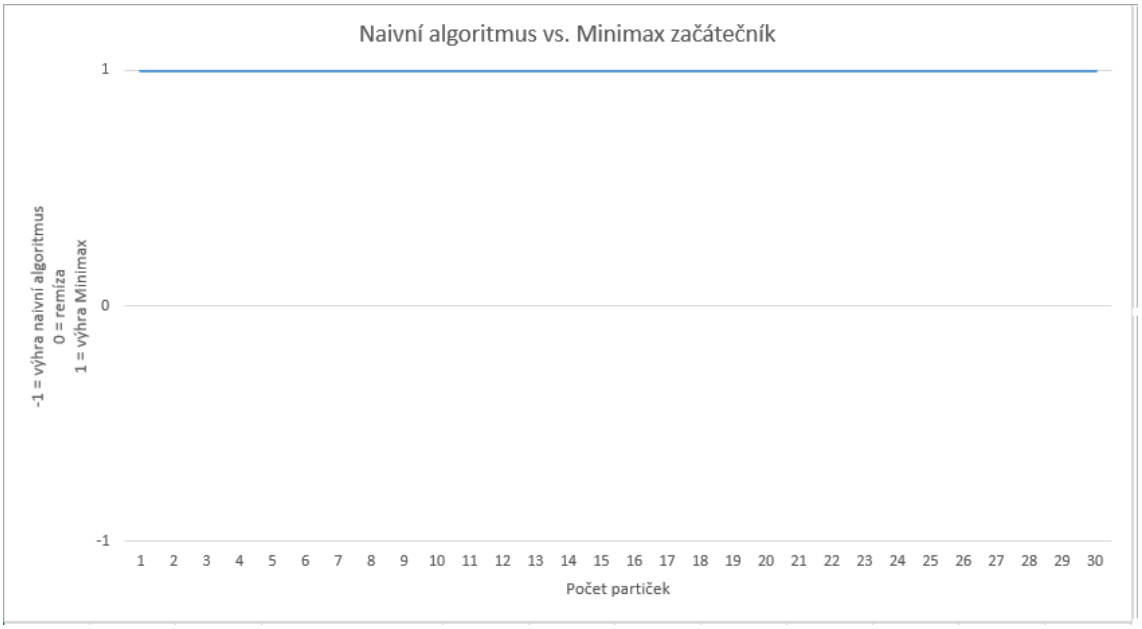
[36] Projekt MONO: *http://www.mono-project.com/Main_Page*

[37] Bitboard: *http://pages.cs.wisc.edu/ psilord/blog/data/chess-pages/index.html*

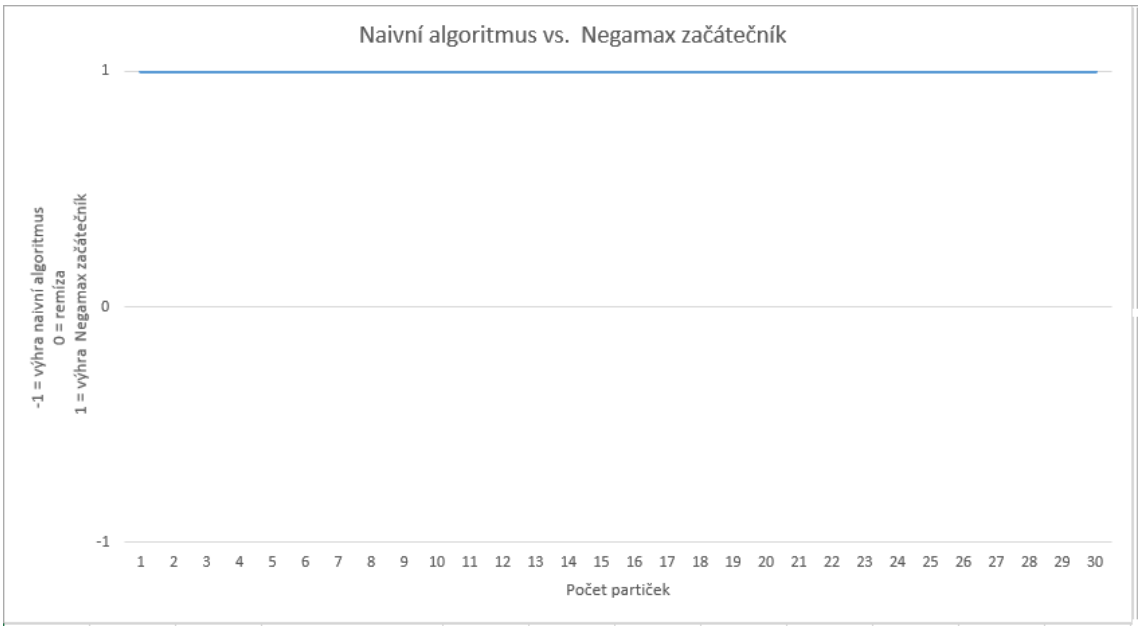
10 Grafy a měření



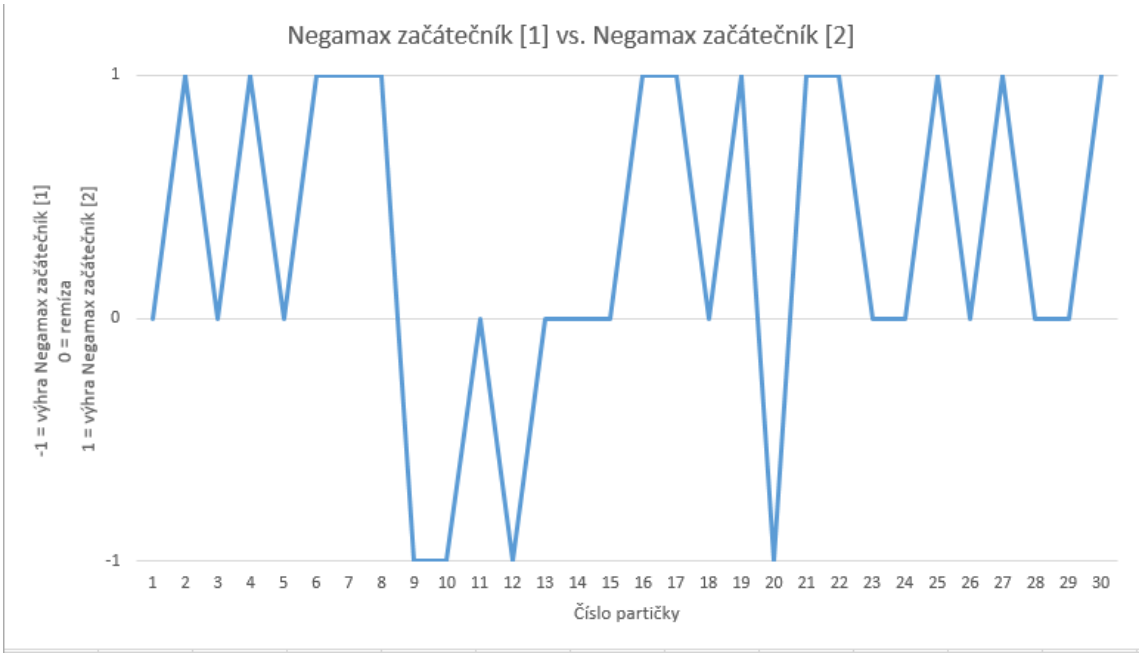
Obrázek 23: Naivní algoritmus vs. Minimax začátečník



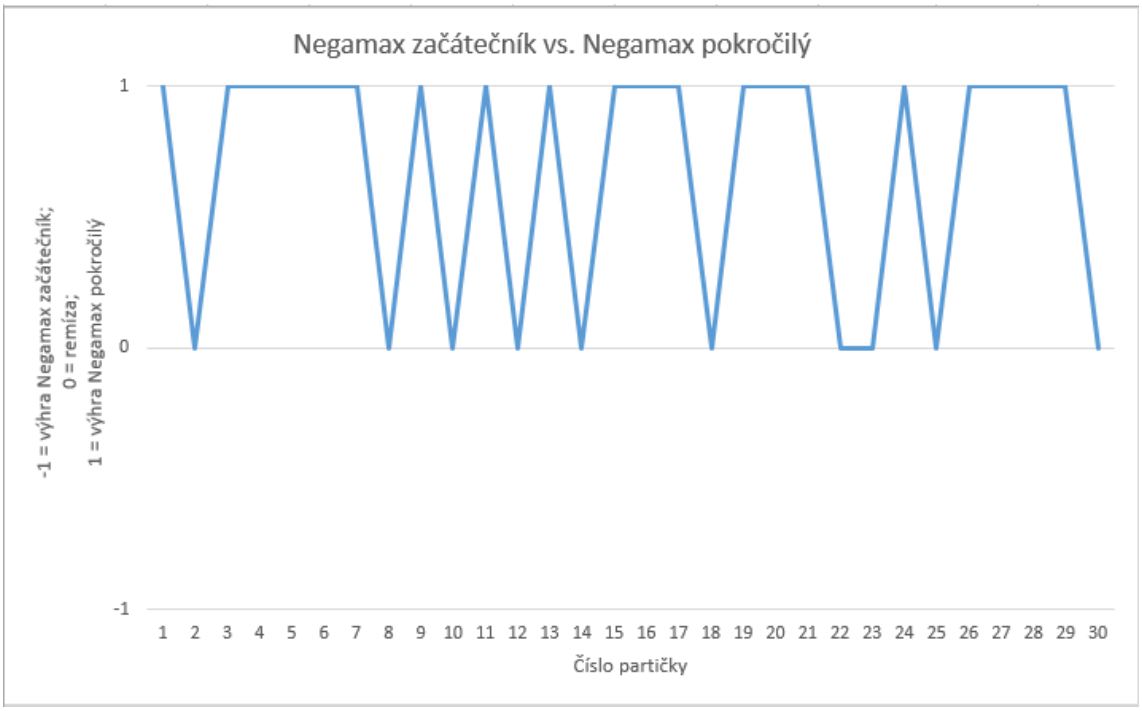
Obrázek 24: Naivní algoritmus vs. Minimax začátečník



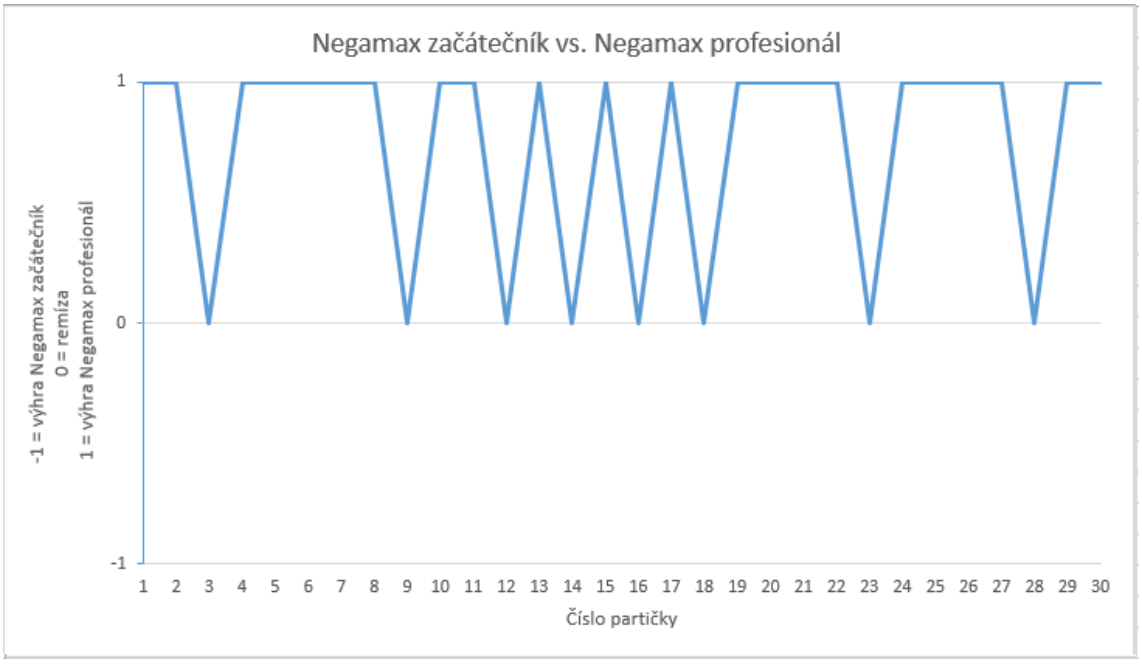
Obrázek 25: Naivní algoritmus vs. Minimax začátečník



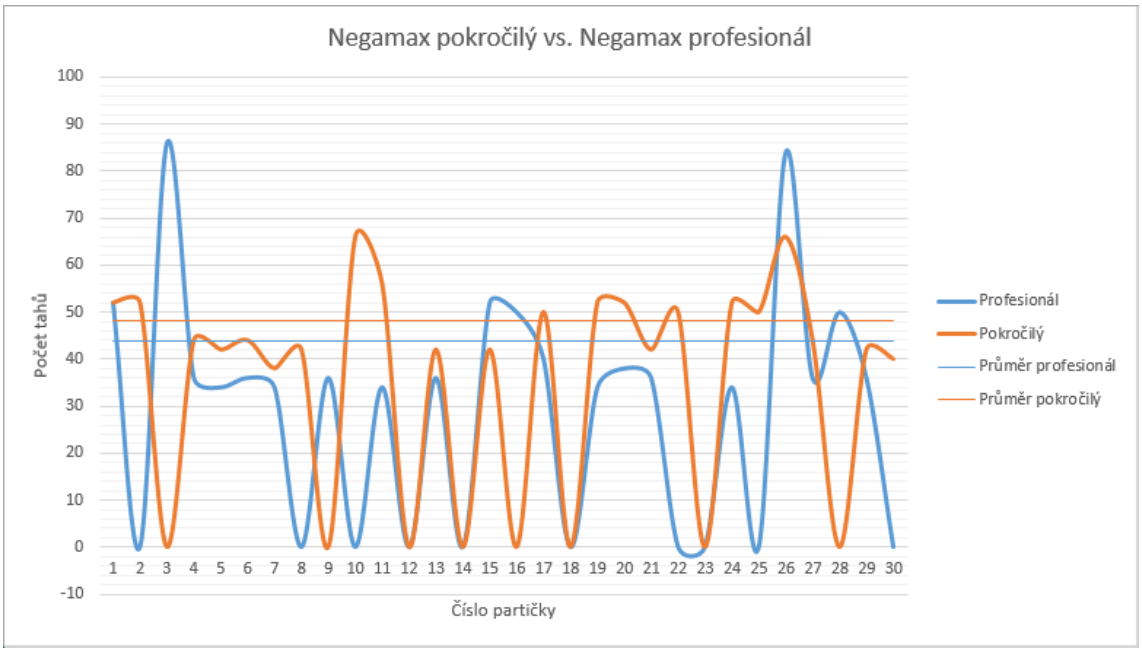
Obrázek 26: Negamax začátečník vs. Negamax začátečník



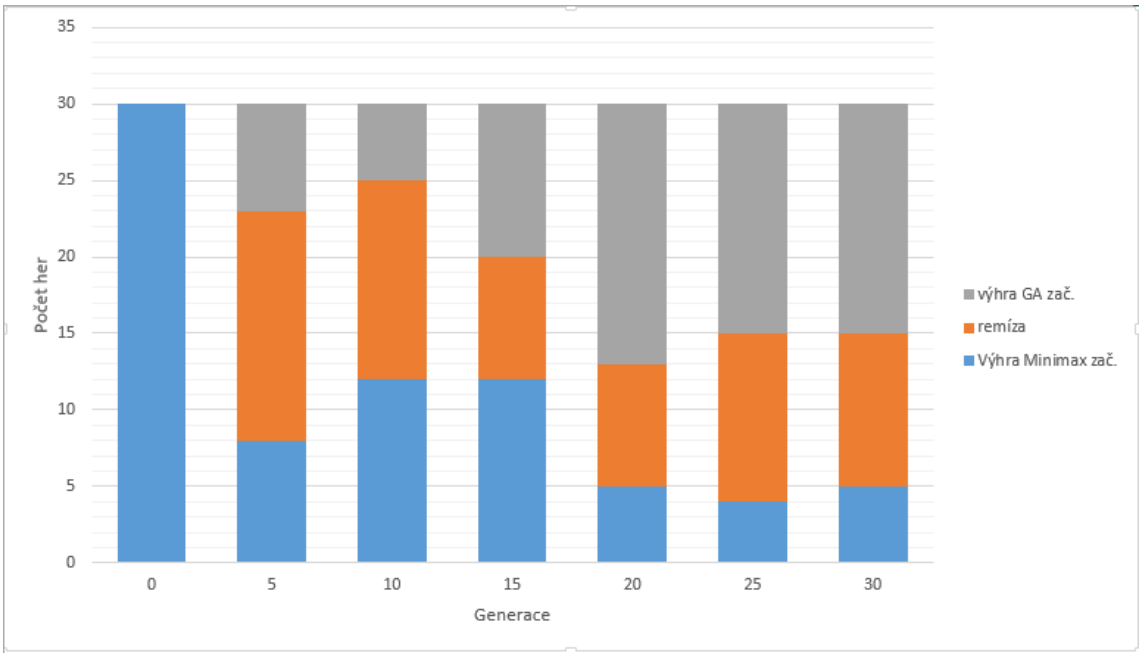
Obrázek 27: Negamax začátečník vs. Negamax pokročilý



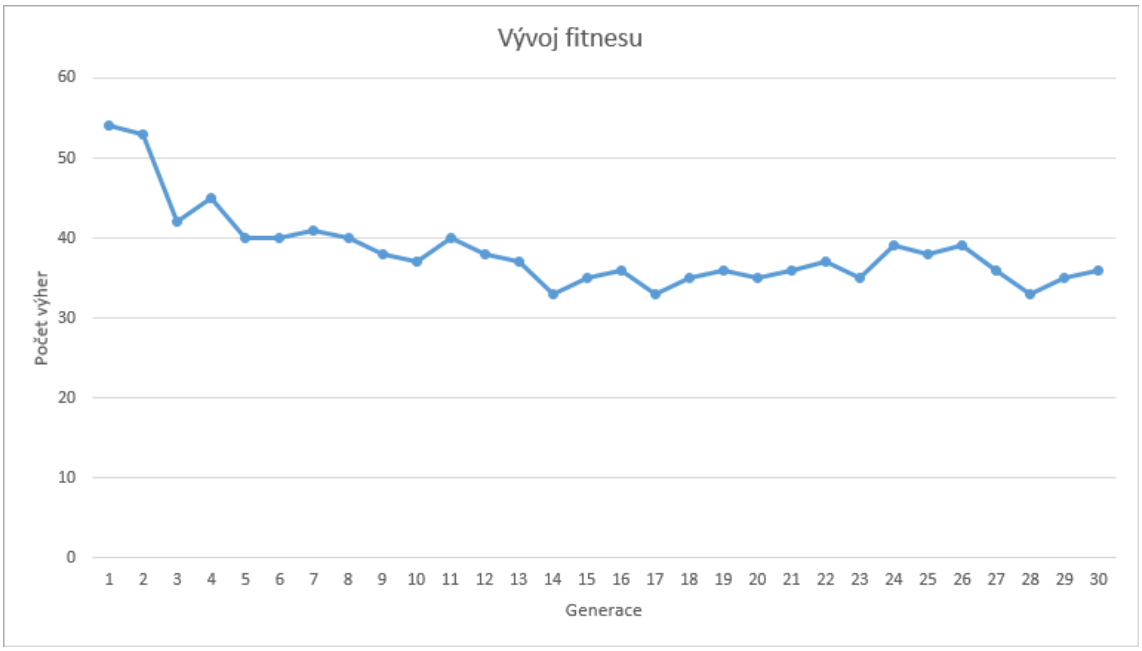
Obrázek 28: Negamax začátečník vs. Negamax profesionál



Obrázek 29: Negamax pokročilý vs. Negamax profesionál



Obrázek 30: GA začátečník vs. Minimax začátečník



Obrázek 31: Vývoj fitnesu